robotron

DCP

SOFTWARE
Dokumentation

Anleitung für den Assemblerprogrammierer
Teil II – Heft 2

Anleitung
fuer
den Assemblerprogrammierer
Teil 2 - Heft 2

VEB Robotron Buchungsmaschinenwerk
Karl-Marx-Stadt, 1987

Die vorliegende 1. Auflage der Dokumentation "Anleitung fuer den Assemblerprogrammierer" unter DCP 3.2 entspricht dem Stand vom 30.6.87 und unterliegt nicht dem Aenderungsdienst.

Nachdruck, jegliche Vervielfaeltigung oder Auszuege daraus sind unzulaessig.

Die Dokumentation wurde durch ein Kollektiv des

VEB Robotron Buchungsmaschinenwerk Karl-Marx-Stadt

erarbeitet.

Bitte senden Sie uns Ihre Hinweise, Kritiken, Wuensche oder Forderungen zur Dokumentation zu.

VEB Robotron Buchungsmaschinenwerk
Karl-Marx-Stadt
PSF 129
Karl-Marx-Stadt
9010

Die "Anleitung fuer den Assemblerprogrammierer" besteht aus zwei Teilen

Teil 1 enthaelt:

I.	CPU - Befehlsbeschreibung	Heft 1
II.	Assembler (MASM)	Heft 2

Teil 2 enthaelt:

III.	Editoren (EDLIN, BE)	Heft 1
IV.	Bibliotheksverwalter (LIB)	Heft 1
V.	Binder (LINK)	Heft 2
VI.	Debugger (SYMDEB)	Heft 2
VII.	MAKE	Heft 2

I N H A L T S V E R Z E I C H N I S

		Seite
V.	**BINDER LINK**	6
1.	Einleitung	6
2.	Bedienung von LINK	6
2.1.	Nutzung der Bedienerfuehrung durch das Programm	6
2.2.	Verwendung der Befehlszeile zum Spezifizieren der LINK-Dateien	8
2.3.	Verwendung einer Antwortdatei zum Spezifizieren der LINK-Dateien	10
2.4.	Suchpfade in Bibliotheken	12
2.5.	Die Listdatei	13
2.6.	Die temporaere Diskettendatei VM.TMP	14
3.	Verwendung der LINK-Schalter	15
3. 1.	Anzeigen der Schalterliste	16
3. 2.	Pause zum Wechseln von Disketten	17
3. 3.	Packen der ausfuehrbaren Datei	17
3. 4.	Erstellen der Liste aller Eintrittspunkte	18
3. 5.	Kopieren der Zeilennummern in die Listdatei	18
3. 6.	Unterscheiden Gross- und Kleinbuchstaben	19
3. 7.	Ignorieren von Standardbibliotheken	20
3. 8.	Setzen der Stackgroesse	20
3. 9.	Setzen der maximalen Anzahl von Leerzeichen	21
3.10.	Setzen einer hohen Startadresse	21
3.11.	Zuweisung einer Datengruppe	22
3.12.	Entfernen von Gruppen aus einem Programm	22
3.13.	Setzen Ueberlagerungsinterrupt	23
3.14.	Setzen der Maximalzahl von Segmenten	23
3.15.	Verwenden der DCP-Segmentordnung	24
4.	Arbeitsweise von LINK	24
4.1.	Reihenfolge der Segmente	25
4.2.	Rahmennummer	25
4.3.	Anordnung der Segmente	25
4.4.	Kombinierte Segmente	26
4.5.	Gruppen	26
4.6.	Finden nicht aufgeloester Bezugnahmen	27
5.	Fehlermeldungen von LINK	28
VI.	**DEBUGGER SYMDEB**	36
1.	Einfuehrung	36
2.	Symbolisches Testen	36
3.	MAPSYM	36
4.	Start von SYMDEB	37

		Seite
4.1.	Start mit nur einer ausfuehrbaren Datei	38
4.2.	Start fuer symbolisches Testen	38
4.3.	Programmargumente	39
4.4	Start ohne Datei	40
5.	SYMDEB-Optionen	40
5.1.	Interaktive Unterbrechungstaste	40
5.2.	Bildschirm-Anzeigewechsel	41
5.3.	Start Kommandos	41
6.	Kommando-Parameter	42
6.1.	Symbole	42
6.2.	Zahlen	43
6.3.	Adressen	43
6.4.	Adress-Bereiche	44
6.5.	Objekt-Bereiche	44
6.6	Zeichenfolgen	45
6.7.	Ausdruecke	45
7.	SYMDEB-Kommandos	46
7.1.	Assemblieren	47
7.2.	Unterbrechungspunkte	50
7.2.1.	Unterbrechungspunkte setzen	50
7.2.2.	Unterbrechungspunkte loeschen	51
7.2.3.	Unterbrechungspunkte entaktivieren	51
7.2.4.	Unterbrechungspunkte aktivieren	52
7.2.5.	Unterbrechungspunkte listen	52
7.3.	Kommentar	53
7.4.	Vergleichen	53
7.5.	Anzeige	54
7.6.	Speicheranzeige	54
7.6.1.	Speicheranzeige Byte	55
7.6.2.	Speicheranzeige ASCII	55
7.6.3.	Speicheranzeige Worte	56
7.6.4.	Speicheranzeige Doppelworte	56
7.6.5.	Speicheranzeige Gleitkommazahlen	57
7.7	Tastatureingabe	57
7.7.1.	Eingabe Byte	58
7.7.2.	Eingabe ASCII	59
7.7.3	Eingabe Wort	59
7.7.4.	Eingabe Doppelwort	59
7.7.5	Eingabe Gleitkommazahl	60
7.8.	Anzeige Symboltabelle	60
7.9.	Fuellen	62
7.10.	Echtzeitabarbeitung	62
7.11.	Hexa	63
7.12.	Port-Eingabe	63
7.13.	Laden	64
7.14.	Transport	65
7.15.	Name	65
7.16.	Eroeffnen Symboltabelle	66
7.17.	Port-Ausgabe	67
7.18.	PTrace	67

		Seite
7.19.	Beenden	68
7.20.	Umlenkung	68
7.21.	Register	69
7.22.	Anzeigewechsel	71
7.23.	Suchen	71
7.24.	Shell Escape	72
7.25.	Stack Trace	73
7.26.	Setzen Symbolwert	74
7.27.	Trace	74
7.28.	Reassemblieren	74
7.29.	Schreiben	75
ANHANG	Test von Quellprogrammen mit SYMDEB	76
VII.	**MAKE**	81
1.	Einleitung	81
2.	Anwenden von MAKE	81
2.1.	Erstellen einer MAKE-Beschreibungsdatei	81
2.2.	Starten von MAKE	83
2.3.	MAKE-Optionen	83
2.4.	Anwenden von Makro-Definitionen	84
2.5.	Verschachtelung von Makro-Definitionen	86
2.6.	Anwendung spezieller Makros	86
2.7.	Standardregeln	87
3.	Beispiel fuer das Pflegen eines Programmes	88
4.	Fehleranzeigen	89
5.	Exit-Codes	91

V. BINDER LINK

1. Einleitung

Der Binder (LINK) erstellt ausfuehrbare Programme, die in Objektmoduln, generiert durch den Makroassembler MASM oder durch hoehere Programmiersprachen, vorliegen muessen. Das Programm LINK liefert als Ergebnis eine ausfuehrbare Datei mit der Dateierweiterung .EXE.
Diese Datei kann in der DCP-Befehlszeile durch Eingabe des Dateinamens aufgerufen und abgearbeitet werden.
Um das Programm LINK verwenden zu koennen, muessen vorher ein oder mehrere Objektmoduln erstellt werden. Diese Moduln koennen auch in speziellen Bibliotheken, die mit dem Programm LIB erzeugt wurden, enthalten sein. LINK verbindet Kode und Daten in den Objektmoduln und durchsucht die angegebenen Bibliotheken, um externe Bezugnahmen in Routinen aufzuloesen.
Durch Erzeugung von Verschiebeinformationen kann DCFX das Programm an jede geeignete Speicherposition laden und abarbeiten.
LINK kann Programme bis zu 1 MByte binden.

2. Bedienung von LINK

In diesem Kapitel werden die unterschiedlichen Moeglichkeiten der Bedienung des Programmes naeher erlaeutert.
Es werden 3 Moeglichkeiten in der Bedienung unterschieden:

- durch Beantworten einer Serie von Anzeigen auf dem Bildschirm (Bedienerfuehrung durch das Programm)
- in einer Befehlszeile von DCP
- durch Nutzung einer vorbereiteten Antwortdatei

Alle 3 Methoden koennen auch kombiniert angewendet werden. Der Binder kann waehrend der Arbeit jederzeit durch das Eingeben von CTRL-C verlassen werden.

2.1. Nutzung der Bedienerfuehrung durch das Programm

In der Befehlszeile von DCP wird LINK <ENTER> eingegeben. Damit wird das Programm in den Speicher geladen. Es erfolgt die Anzeige aller notwendigen Informationen auf dem Bildschirm.
Folgende Schritte sind durch den Bediener auszufuehren:

- Object Modules [.OBJ]:

Diese Anzeige fordert zur Eingabe der Objektmoduln auf. Es erfolgt die Eingabe der Dateibezeichnungen der zu bindenden Moduln. Wird keine Dateierweiterung mit eingegeben, so verwendet LINK die Standarddateierweiterung .OBJ.
Sind mehrere Moduln zu binden, so sind sie mit Leerzeichen oder dem Zeichen "+" zu trennen.
Wird zur Eingabe der Moduln mehr als eine Zeile benoetigt, dann muss als letztes Zeichen in der Zeile ein "+" stehen. Anschliessend wird <ENTER> betaetigt. Die Anzeige "Objekt Modules

[.OBJ]: " erscheint erneut und es koennen weitere Moduln eingegeben werden.
Sind alle Objektmoduln eingegeben, dann ist die Eingabe mit <ENTER> abzuschliessen.
Auf dem Bildschirm erscheint die naechste Anzeige.

- Run File [filename.EXE]:

Es gibt zwei Antwortmoeglichkeiten:

a) Eingabe von <ENTER>:
   Als Dateiname wird der Name des ersten Objektmoduls verwendet und die Dateierweiterung .EXE eingefuegt.

b) Eingabe Dateiname und <ENTER>:
   Die ausfuehrbare Datei erhaelt diesen Namen und die Erweiterung .EXE.

- Es folgt die Anzeige:

  List File [NUL.MAP]:

Soll eine Listdatei erstellt werden, ist der Name der Datei einzugeben und <ENTER> zu bedienen. LINK erzeugt standardmaessig die Dateierweiterung .MAP.
Wird keine Listdatei gewuenscht, so wird die Anzeige nur durch <ENTER> quittiert.

- Die naechste Anzeige lautet:

  Libraries [.LIB]:

Es sind die Namen der Bbliotheken einzugeben, die zum Erstellen der ausfuehrbaren Datei notwendig sind. Wird mehr als ein Name eingegeben, sind diese durch Leerzeichen oder Plus (+) zu trennen. Wird keine Dateierweiterung angegeben, so setzt der Binder .LIB als Standard voraus. Sollen mehr Bibliotheken eingegeben werden, als in eine Zeile passen, ist als letztes Zeichen der Zeile ein Plus (+) einzugeben und <ENTER> zu betaetigen. Damit erscheint die Anzeige "Libraries [.LIB]:" auf der naechsten Zeile noch einmal, und es koennen weitere Bibliotheken eingegeben werden. Werden fuer den Bindelauf keine Bibliotheken benoetigt, dann ist die Anzeige nur mit <ENTER> zu quittieren.
Danach erstellt LINK die ausfuehrbare Datei.
Bei der Eingabe der Dateinamen ist zu beachten, dass fuer jede Datei, die sich nicht im aktuellen Laufwerk und im aktuellen Verzeichnis befindet, ein Pfad einzugeben ist. Die LINK-Schalter koennen nach den Dateibezeichnungen nach jeder Anzeige eingegeben werden.
Findet der Binder einen Objektmodul nicht, so gibt er eine Fehlermeldung aus und wartet, dass die Diskette gewechselt wird, falls dies notwendig ist. Nach jeder Anzeige ist es moeglich, die fehlenden Dateibezeichnungen in Form einer Befehlszeile einzugeben, wie es im Punkt 2.2. beschrieben wird. Weiterhin kann durch die Eingabe eines Semikolon (;) nach einer beliebigen

Anzeige LINK veranlasst werden, fuer alle uebrigen Eingaben die Standardantworten zu generieren. Wird ein Semikolon nach der Anzeige "Object Modules" eingegeben, muss vorher mindestens eine Objektmodulbezeichnung eingegeben worden sein. Erkennt der Binder ein Semikolon, dann erzeugt er selbst die Standardantworten und Dateien ohne Anzeige auf dem Bildschirm. Weiterhin ist die Eingabe von Kommas (,) moeglich, um verschiedene Dateien zu erstellen.

Beispiel:

```
LINK
Object Modules [.OBJ]: mod1+mod2+mod3
Object Modules [.OBJ]: mod4+modup/PAUSE
Run File [mod1.exe]  :
List File [NUL.MAP]  : listdat
Libraries [.LIB]     : b:\bib\updat
```

Dieses Beispiel bindet die Objektmoduln mod1.obj, mod2.obj, mod3.obj, mod4.obj und modup.obj. Es wird die Bibliothek updat.lib im Laufwerk B und dem Verzeichnis bib nach Routinen und Daten durchsucht, die im Programm verwendet werden.
Anschliessend werden die ausfuehrbare Datei mod1.exe, sowie die Listdatei listdat.map erstellt und auf der Diskette im aktuellen Laufwerk abgespeichert.
Der Schalter /PAUSE in der Zeile der Objektmoduln veranlasst den Binder anzuhalten, weil Disketten gewechselt werden sollen. Danach wird die ausfuehrbare Datei erstellt (siehe Punkt 3.2.).

2.2. Verwendung der Befehlszeile zum Spezifizieren der LINK-Dateien

Die ausfuehrbare Datei kann auch durch Eingabe in der Befehlszeile erstellt werden.
Die Befehlszeile besitzt folgende allgemeine Form:

LINK <objektmodulbezeichnungen>[;[<bezeichnung der ausfuehrbaren datei>][, [<listdateibezeichnung>] [, [<bibliotheksdateibezeichnungen>]]]] [</schalter>] [;]

- <objektmodulbezeichnungen>:

Sie enthalten den oder die Namen der Objektmoduln, die eingegeben werden sollen. Sie muessen durch MASM oder Compiler hoeherer Programmiersprachen erstellt worden sein.
Der Binder fordert als Eingabe mindestens einen Objektmodul. Wird keine Dateierweiterung angegeben, setzt LINK standardmaessig .OBJ ein.

Alle folgenden Eingaben sind optional.

- <bezeichnung der ausfuehrbaren datei>:

Es kann hier eine Dateibezeichnung fuer die ausfuehrbare Datei

eingegeben werden. Wird die Eingabe freigelassen, so erhaelt die ausfuehrbare Datei den Namen des ersten in der Liste stehenden Objektmoduls und die Dateierweiterung .EXE.

- <listdateibezeichnung>:

Eine Eingabe einer Dateibezeichnung ist nur notwendig, wenn eine entsprechende Listdatei durch den Binder erzeugt werden soll. Mit dem Schalter /MAP oder /LINENUMBERS kann ebenfalls eine Listdatei erzeugt werden, ohne dass in der Befehlszeile eine derartige Datei spezifiziert wurde.

- <bibliotheksdateibezeichnungen>:

Wenn Bibliotheken zum Binden verwendet werden sollen, so muessen die Dateibezeichnungen an dieser Stelle eingegeben werden. Wird keine Dateierweiterung angegeben, so wird von LINK standardmaessig .LIB verwendet. Die Eingabe ist nur notwendig, wenn Bibliotheken fuer den Bindelauf benoetigt werden.

- </schalter>:

Die Schalter steuern die Operationen des Binders. Sie sind im Gliederungspunkt 3.3. aufgefuehrt. Sie koennen an jeder beliebigen Stelle innerhalb der Befehlszeile stehen.

Die Kommas werden zur Trennung der einzelnen unterschiedlichen Dateien gesetzt. Sie werden auch an den Stellen gefordert, wo kein Dateiname eingegeben wurde. Semikolon kann nach der Eingabe der Objektmoduln eingegeben werden, um die Befehlszeile vorzeitig zu beenden.
Wird Semikolon gleich nach den Objektmoduln geschrieben, dann setzt LINK standardmaessig den Namen des ersten Moduls fuer die ausfuehrbare Datei ein, es wird keine Listdatei erstellt und keine Bibliothekdatei verwendet.
Wenn nicht alle Eingaben in der Befehlszeile getaetigt wurden und sie nicht mit Semikolon abgeschlossen wurden, dann bringt der Binder die weiteren Anzeigen, wie im Gliederungspunkt 2.1. beschrieben.
Soll mehr als ein Objektmodul oder Bibliothek gebunden werden, sind sie mit Leerzeichen oder Plus (+) zu trennen. Die entsprechende Datei wird dem aktuellen Laufwerk und Verzeichnis zugewiesen, falls kein Laufwerk oder Verzeichnis spezifiziert wurde. Das zugewiesene Laufwerk oder Verzeichnis gilt dabei immer nur fuer die darauffolgende Datei. Die Position jeder Datei muss extra spezifiziert werden.
Weiterhin ist zu beachten, wenn Objektmoduln, die mit einem Compiler von hoeheren Programmiersprachen erzeugt wurden, Ueberlagerungsstrukturen enthalten, dann muessen die Ueberlagerungsmoduln in runde Klammern geschrieben werden. MASM kann keine solche Ueberlagerungsmoduln erzeugen.

Beispiele:

- **LINK datei1.obj, datei1.exe, datei1.map, upbib.lib**

Dieses Beispiel hat die gleiche Wirkung wie die folgende Zeile:

```
LINK datei1,,,upbib
```

Der Dateiname des Objektmoduls datei1 wird verwendet, um die ausfuehrbare Datei datei1.exe zu erstellen. LINK durchsucht die Bibliothek upbib nach Routinen und Variablen, die im Programm verwendet werden. Weiterhin wird die Listdatei datei1.map erstellt, deren Inhalt eine Liste der Segmente und Gruppen des Programms enthaelt.

- `LINK datmod1 + prog1,b:prog1,\map\prog1;`

Der Binder nimmt die Objektmoduln datmod1.obj und prog1.obj vom aktuellen Laufwerk und erstellt auf der Diskette im Laufwerk B die ausfuehrbare Datei prog1.exe. Die Listdatei prog1.map wird im aktuellen Laufwerk im Verzeichnis map abgelegt. Die Kommandozeile wird mit Semikolon abgebrochen, d.h. es werden keine Bibliotheken fuer den Bindelauf verwendet.

- `LINK mod1 mod2 mod3 modup/PAUSE,,list,b:bib\upbib`

Es werden die Objektmoduln mod1.obj, mod2.obj,mod3.obj und modup.obj gebunden. Weiterhin durchsucht LINK die Bibliothek upbib.lib im Laufwerk B, Verzeichnis bib nach Routinen und Daten, die im Programm benoetigt werden. Sie werden mit eingebunden. Anschliessend veranlasst der Schalter /PAUSE den Binder zu einem Halt. Damit ist das Wechseln der Diskette moeglich, bevor die ausfuehrbare Datei ausgegeben wird (siehe Gliederungspunkt 3.2.). Danach wird die ausfuehrbare Datei mod1.exe und die Listdatei List.map ausgegeben.

2.3. Verwendung einer Antwortdatei zum Spezifizieren der LINK-Dateien

Die Antwortdatei ist eine Datei, die alle notwendigen Eingaben fuer den Bindelauf enthaelt. Sie muss vorher erzeugt werden. Der einfachste Weg, eine Antwortdatei zu verwenden, besteht in einer Befehlszeile in der Form:

```
LINK @<antwortdatei>
```

Eine Antwortdatei kann auch nach jeder Anzeige oder in jeder Position in der Befehlszeile aufgerufen werden. Die Antworten von der Antwortdatei muessen exakt so sein, als wuerden sie bei jeder Anzeige oder in der Befehlszeile eingegeben. Soll eine Antwortdatei aufgerufen werden, dann muss der Dateiname mit dem Zeichen @ beginnen. Befindet sich die Datei in einem anderen Laufwerk oder Verzeichnis, so muss der entsprechende Pfad mit eingegeben werden.
Der Name der Antwortdatei ist dem Nutzer ueberlassen.

Eine Antwortdatei hat folgende allgemeine Form:

```
[bezeichnung der zu bindenden objektmoduln]
[bezeichnung der ausfuehrbaren datei]
```

[dateibezeichnung der listdatei]
[dateibezeichnung der bibliotheken]

Elemente, die bereits auf Anzeigen oder in der Befehlszeile eingegeben wurden, koennen weggelassen werden.
Jede Gruppe von Dateibezeichnungen muss in einer separaten Zeile beginnen. Sind mehr Dateibezeichnungen vorhanden, als in eine Zeile passen, dann koennen weitere Dateibezeichnungen in der naechsten Zeile eingegeben werden, wenn in der vorherigen Zeile als letztes Zeichen ein Plus (+) eingegeben wurde.
Soll fuer eine Gruppe keine Dateibezeichnung eingegeben werden, dann ist eine Leerzeile einzugeben. Die Eingabe von Schaltern ist in jeder Zeile moeglich. Ein Semikolon ist in jeder Zeile der Antwortdatei moeglich. Erkennt LINK das Semikolon, werden automatisch die Standarddateibezeichnungen fuer alle Dateien erzeugt, die nicht in der Antwortdatei benannt wurden.
Der Rest der Antwortdatei (nach dem Semikolon) wird dann durch den Binder ignoriert.
Wird eine ausfuehrbare Datei mit einer Antwortdatei erstellt, zeigt der Binder jede Antwort von der Datei auf dem Bildschirm an. Sind in der Antwortdatei die notwendigen Dateien nicht enthalten, wartet LINK auf die manuelle Eingabe dieser Dateibezeichnungen.
Eine Antwortdatei ist immer mit Semikolon oder <ENTER> abzuschliessen. Fehlt das letzte <ENTER> am Ende der Antwortdatei, zeigt der Binder die letzte Zeile dieser Datei auf dem Bildschirm an und wartet, dass <ENTER> betaetigt wird.

Beispiele:

- Antwortdatei ANTW

```
mod1 mod2 mod3 mod4/PAUSE

listdat
b:\bib\upprog
```

Diese Antwortdatei fordert den Binder auf, die Objektmoduln mod1, mod2, mod3 und mod4 zu binden. Danach haelt der Binder an, um einen Diskettenwechsel zu ermoeglichen, bevor die ausfuehrbare Datei mod1.exe erzeugt wird.
Weiterhin erstellt LINK die Listdatei listdat.map und durchsucht die Bibliotheken upprog.lib im Laufwerk B im Verzeichnis \bib.
Aufruf der Antwortdatei erfolgt mit: LINK @antw

- In diesem Beispiel werden alle 3 Moeglichkeiten kombiniert angewendet. Dazu existiert die Antwortdatei bib1, die folgende Zeile enthaelt:

```
bib1+bib2+bib3+bib4
```

Nun wird LINK mit einem Teil der Befehlszeile gestartet:

```
LINK omod1 omod2
```

Der Binder nimmt die Objektmoduln omod1.obj und omod2.obj an und bringt die Anzeige fuer die naechste Datei, da die Befehlszeile nicht mit Semikolon beendet wurde:

```
Run File  [omod1.exe]:aprog
List File [NUL.MAP]:
Libraries [.LIB]    : @bibl
```

Durch die Eingabe von aprog wird die ausfuehrbare Datei aprog.exe erzeugt. Fuer die Listdatei wurde nur <ENTER> eingegeben, so dass diese Datei nicht erzeugt wird. Die Eingabe @bibl veranlasst den Binder, die Antwortdatei bibl abzuarbeiten und die dort enthaltenen Bibliotheken in den Bindevorgang mit einzubeziehen.

2.4. Suchpfade in Bibliotheken

Mit dem Programm LINK ist es moeglich Verzeichnisse und Laufwerke fuer Bibliotheken zu suchen, die in einem Befehl in Suchpfaden mit Bibliotheksnamen spezifiziert wurden bzw. durch Zuweisung von Suchpfaden in der Umgebungsvariablen LIB, bevor LINK aktiviert wird. Umgebungsvariable sind unter dem Kommando SET im DCP-Handbuch erklaert.
Ein Suchpfad ist eine Pfadspezifizierung eines Verzeichnisses oder eines Laufwerkes. Suchpfade koennen gemeinsam mit dem Bibliotheksnamen in der LINK-Befehlszeile oder als Antwort zu der "Libraries"-Anzeige eingegeben werden.
Es koennen maximal 16 Suchpfade spezifiziert werden. Suchpfade koennen ebenfalls zur LIB-Umgebungsvariablen zugewiesen werden. Dazu muss das DCPX-Kommando SET verwendet werden. Die Suchpfade sind durch Semikolon zu trennen.
Wenn ein Laufwerks- oder Verzeichnisname in der Dateibezeichnung fuer eine Bibliothek in der LINK-Befehlszeile enthalten ist, sucht der Binder nur nach diesem. Wurde kein Laufwerks- oder Verzeichnisname eingegeben, sucht LINK in der folgenden Reihenfolge nach den Bibliotheksdateien:

1. Der Binder durchsucht das aktuelle Laufwerk und Verzeichnis
2. Wird die Bibliothek nicht gefunden, sind aber ein oder mehrere Suchpfade in der Befehlszeile angegeben, durchsucht der Binder die spezifizierten Pfade in der angegebenen Reihenfolge.
3. Ist die Bibliothek immer noch nicht gefunden und ein Suchpfad ist mit der LIB-Umgebungsvariablen vorgegeben, durchsucht ihn der Binder.
4. Wurde die Bibliothek immer noch nicht gefunden, so wird eine Fehlermeldung ausgegeben.

Beispiele:

- **LINK datei,,datei,A:\altbib\upprog.LIB+common+B:+D:\neubib**

Der Binder durchsucht nur das Verzeichnis altbib im Laufwerk A, um die Bibliothek upprog.LIB zu finden. Bei common.LIB wird das

aktuelle Verzeichnis im aktuellen Laufwerk, sowie das aktuelle Verzeichnis im Laufwerk B und zuletzt das Verzeichnis neubib im Laufwerk D durchsucht.

```
- SET LIB=C:\lib;D:systemlib
  LINK datei,,datei.map,upprog+upmath
```

Der Binder durchsucht das aktuelle Verzeichnis, das Verzeichnis \lib im Laufwerk C und das Verzeichnis \system\lib im Laufwerk D, um die Bibliotheken upprog.lib und upmath.lib zu finden.

2.5. Die Listdatei

Die Listdatei mit der Dateierweiterung .MAP, listet alle Namen, Ladeadressen und Laengen aller Segmente eines Programmes auf. Es werden die Namen und Ladeadressen von Gruppen im Programm, die Startadresse und Fehlermitteilungen, sofern sie auftreten, aufgelistet. Bei Anwendung des Schalters /MAP in der LINK-Befehlszeile werden die Namen und Adressen aller Eintritts-(Public-) Symbole aufgelistet.
Die Segmentinformation hat folgende Form:

Start	Stop	Length	Name	Class
00000H	0165CH	0165CH	TEXT	CODE
01660H	01799H	00139H	DATA	DATA

Die Start- und Stop-Spalten zeigen die 20-Bit-Adressen (hexadezimal) des ersten und letzten Bytes in jedem Segment. Diese Adressen werden relativ zum Beginn des Lademoduls, der mit der Adresse 00000H beginnt, angegeben.
Das Betriebssystem legt die Startadresse dann fest, wenn das Programm geladen wird.
Die Spalte Length gibt die Groesse des Segments in Byte (hexadezimal) an, waehrend die Spalte Name den Namen des Segments angibt. Die Spalte Class enthaelt den Klassennamen des Segments.

Die Gruppeninformation hat folgende Form:

Origin	Group
0000:0	CGROUP
0166:0	DGROUP

In diesem Beispiel ist CGROUP der Name einer Kodegruppe und DGROUP ist der Name einer Datengruppe.
Am Ende der Listdatei teilt der Binder den Programmeintrittspunkt mit. Wurde in der LINK-Befehlszeile der Schalter /MAP eingegeben, dann haengt der Binder eine Liste aller PUBLIC-Symbole an.

Die Symbole werden zweimal aufgelistet:

a) in alphabetischer Reihenfolge
b) in der Reihenfolge ihrer Adressen

Die Form der Liste wird im folgenden Beispiel anschaulich gemacht:

```
     Address             Publics by Name

     0000:160A           BPOINT
     0000:01AC           CALLUP1
     0000:12C1           CLERR
     0000:16FC           CMOD1
     0153:00B1           FMUL


     Address             Publics by Value

     0000:01AC           CALLUP1
     0000:12C1           CLERR
     0000:160A           BPOINT
     0000:16FC           CMOD1
     0153:00B1           FMUL
```

Die Adressen der Eintrittssymbole sind im Segment:Offset-Format dargestellt. Sie zeigen die Positionen relativ zum Beginn des Lademoduls, der auf die Adresse 0000:0000 festgelegt ist.
Wenn die Schalter /HIGH und /DSALLOCATE (Punkt 3.10 und 3.11) verwendet wurden und das Programm, Kode und Daten zusammen nicht 64 KByte ueberschreiten, dann zeigt die Listdatei die Symbole mit ungewoehnlich grossen Segmentadressen an. Diese Adressen zeigen auf ein Symbol, dessen Position unterhalb der Startadresse des Programmkodes und der -daten liegt.

Beispiel:

```
     EEE0:0A20   PROG1
```

PROG1 ist in diesem Beispiel unterhalb des Programmstarts positioniert. Damit ergibt sich die 20-Bit-Adresse von PROG1: 00920H.

2.6. Die temporaere Diskettendatei VM.TMP

LINK nutzt fuer den Bindevorgang normalerweise den zur Verfuegung stehenden Speicherplatz. Wird jedoch mehr Speicherplatz benoetigt, dann erstellt der Binder die temporaere Datei VM.TMP im aktuellen Verzeichnis. Sobald diese Datei eroeffnet wird, erscheint auf dem Bildschirm folgende Anzeige:

```
VM.TMP has been created
Do not change diskette in drive X
```

X: Laufwerksbuchstabe

Die Ausschrift lautet: VM.TMP wurde eroeffnet. Die Diskette im Laufwerk X darf nicht gewechselt werden, solange der Bindevorgang nicht beendet ist.
Der Schalter /PAUSE kann nicht genutzt werden, wenn diese temporaere Datei eroeffnet wurde. Diese Datei wird automatisch geloescht, wenn LINK mit der Ausgabe der ausfuehrbaren Datei begonnen hat.
Es ist grundsaetzlich zu beachten, dass die Dateibezeichnung VM.TMP nie fuer Nutzerdateien verwendet werden darf! Eroeffnet der Binder die temporaere Datei, werden alle Informationen der Datei gleichen Namens geloescht.

3. Verwendung der LINK-Schalter

Die Schalter des Binders spezifieren und steuern bestimmte Abschnitte des Bindelaufes. Alle Schalter beginnen mit Schraegstrich (/). Die Schalter koennen an beliebiger Stelle in einer LINK-Befehlszeile verwendet werden.

Schalter	Kurzschreibweise	Bedeutung
/HELP	/HE	Alle Schalter werden angezeigt
/PAUSE	/P	Pause im LINK-Lauf zum Wechsel Diskette, bevor die ausfuehrbare Datei geschrieben wird.
/EXEPACK	/E	Packt (verdichtet) die ausfuehrbare Datei.
/MAP	/M	Auflisten aller Eintrittspunkte (Public-Symbole).
/LINENUMBERS	/LI	Kopiert Zeilennummern in die Listdatei. Wirkung nur auf Listdatei.
/NOIGNORECASE	/NOI	Interpretiert Gross- und Kleinbuchstaben in Symbolen als unterschiedliche Buchstaben.
/NODEFAULTLIBRARYSEARCH	/NOD	Ignoriert alle Bibliotheksnamen, die in Objektmoduln auftreten. Angabe nur explizit ueber Befehlszeile moeglich.

Schalter	Kurz-schreibweise	Bedeutung
/STACK: <zahl>	/ST: <zahl>	Setzt Stackgroesse auf den im Schalter vorgegebenen hexadezimalen positiven Zahlenwert.
/CPARMAXALLOC	/C	Setzen der maximalen Anzahl von 16-Byte-Paragraphen.
/HIGH	/H	Setzt hoechstmoegliche Ladeadresse fuer das Programm im Speicher.
/DSALLOCATE	/D	Daten werden im Speichersegment so hoch wie moeglich eingeschrieben. Wird meist in Verbindung mit /HIGH genutzt.
/NOGROUPASSOCIATION	/NOG	Ignorieren von Gruppen in einem Programm.
/OVERLAYINTERRUPTS <zahl>	/O:<zahl>	Setzt Interruptnummer der Ueberlagerungsroutine auf die eingegebene Zahl.
/SEGMENTS:<zahl>	/SE:<zahl>	Setzt Maximalzahl von Segmenten auf die eingegebene Zahl.
/DOSSEG	/DO	Ordnen der Segmente gemaess der DCPX-Segmentordnung.

Die Schalter koennen sowohl in voller Schreibweise, als auch in der Kurzschreibweise eingegeben werden.
Viele der Schalter setzen Werte im DCPX-Programmkopf (header). Deshalb werden diese Schalter besser verstanden, wenn das Wissen vorhanden ist, wie dieser Kopf aufgebaut ist.

3.1. Anzeigen der Schalterliste

Syntax:

/HELP Kurzschreibweise: /HE

Der Schalter veranlasst den Binder, eine Liste aller moeglichen Schalter auf dem Bildschirm anzuzeigen. Es muss auch vorher keine Dateibezeichnung eingegeben zu werden, wenn dieser Schalter verwendet werden soll.
Beispiel: LINK/HELP

3.2. Pause zum Wechseln von Disketten

Syntax:

/PAUSE Kurzschreibweise: /P

Der Schalter /P veranlasst den Binder, den Bindelauf vor der Ausgabe der ausfuehrbaren Datei (.EXE) zu unterbrechen, um Disketten wechseln zu koennen. Dabei erscheint auf dem Bildschirm folgende Anzeige:

```
About to generate .EXE file
Change diskette in drive x and press <ENTER>
```

X ist der Laufwerksname. Die Anzeige erscheint, nachdem der Binder alle Daten von den Objektmoduln und Bibliotheken eingelesen und, wenn spezifiziert, die Listdatei ausgegeben hat .LINK setzt die Arbeit fort, wenn die Taste <ENTER> bedient wurde.
Nachdem der Binder die ausfuehrbare Datei auf die Diskette geschrieben hat, kommt die Aufforderung, dass die vorhergehende Diskette wieder in das Laufwerk x einzulegen ist:

```
Please replace original diskette
in drive x and press <ENTER>
```

Beim Anwenden des Schalters /P ist Folgendes zu beachten:

- Es darf keine Diskette gewechselt werden, wenn der Binder bereits die temporaere Datei VM.TMP eroeffnet hat.
- Erscheint die Anzeige zum Erstellen von VM.TMP auf dem Bildschirm und es war der Schalter /P spezifiert, muss CTRL-C bedient werden, um den Bindelauf abzubrechen.
  Die Diskette muss so regeneriert werden, dass sowohl die temporaere Datei als auch die zu erstellende Datei auf die gleiche Diskette passen.
  Danach kann der LINK-Lauf erneut gestartet werden.

Beispiel:

```
LINK prog1/PAUSE,prog,,\bib\upprog <ENTER>
```

Das Kommando /PAUSE veranlasst den Binder zu einem Halt, bevor die ausfuehrbare Datei prog.exe generiert wird. Nach der Generierung dieser Datei haelt LINK erneut an, damit die Originaldiskette wieder eingelegt werden kann.

3.3. Packen der ausfuehrbaren Datei

Syntax:

/EXEPACK Kurzschreibweise: /E

Der Schalter /EXEPACK veranlasst den Binder, Folgen sich wiederholender Bytes (meistens Nullen) zu annullieren und die Ladezeit-Verschiebungs-Tabelle zu optimieren, bevor die ausfuehrbare

Datei erzeugt wird. Die ausfuehrbaren Dateien werden dadurch kuerzer und sind somit auch schneller zu laden. Der symbolische Debugger (SYMDEB) kann aber fuer gepackte Dateien nicht verwendet werden.
Der Schalter /E fuehrt jedoch nicht immer zu mehr Platz auf der Diskette. Manchmal kann sich die Datei auch vergroessern. Programme, die eine grosse Zahl Ladezeit-Verschiebungen (etwa 500 und mehr) und viele sich wiederholende Zeichen haben, koennen kuerzer sein, als die gepackten.
Um herauszufinden, ob das Programm den Schalter /E benoetigt, ist es guenstig, beide Varianten (mit und ohne /E-Schalter) zu probieren und die Resultate zu vergleichen.

Beispiel:

```
LINK programm /E;
```

Das Beispiel generiert eine gepackte Version der Datei programm.exe.

3.4. Erstellen der Liste aller Eintrittspunkte

Syntax:

/MAP Kurzschreibweise: /M

Der Schalter /MAP veranlasst den Binder die Liste aller Eintrittspunkte (PUBLIC-Symbole), die im Programm definiert wurden, auszugeben. Die Liste wird auf die Datei mit der Dateierweiterung .MAP geschrieben, die der Binder erstellt. Das Format der Listdatei wurde im Gliederungspunkt 2.5. beschrieben.
Der Schalter /M ist fuer das Testen mit dem Programm SYMDEB notwendig (siehe unter 2. in der Beschreibung von SYMDEB).
Wurde keine Listdatei im LINK-Befehl spezifiert, kann mit dem Schalter /M eine solche erstellt werden. Der Binder gibt der Listdatei dann den Namen des ersten Objektmoduls und die Standarddateierweiterung .MAP.

Beispiel:

```
LINK datei,,/MAP;
```

Dieses Kommando erzeugt die Listdatei datei.map, die alle Eintrittspunkte des Programms enthaelt.

3.5. Kopieren der Zeilennummern in die Listdatei

Syntax:

/LINENUMBERS Kurzschreibweise: /LI

Der Schalter /LI veranlasst den Binder, die Startadresse jeder Programmquellzeile in die Listdatei zu uebernehmen. Die Startadresse ist die Adresse des ersten Befehls, der in der entspre-

chenden Quellzeile steht.
Das Programm MAPSYM kopiert Zeilennummern zu einer Symboldatei, die mit SYMDEB verwendet werden kann.
Der Binder kopiert diese Zeilennummern nur, wenn eine Listdatei in der LINK-Befehlszeile angegeben wurde und wenn der angegebene Objektmodul Informationen ueber Zeilennummern enthaelt. Diese Zeilennummern sind in einigen Compilern hoeherer Programmiersprachen enthalten. MASM uebernimmt keine Informationen ueber Zeilennummern in den Objektmoduln. Besitzt ein Objektmodul keine Zeilennummerninformation, ignoriert der Binder den Schalter /LI. Eine Anwendung beim Binden von Objektmoduln, die ausschliesslich von MASM erzeugt werden, ist sinnlos.
Wird im LINK-Befehl keine Listdatei spezifiert, kann der Schalter /LI zum Erstellen einer Listdatei genutzt werden. Der Schalter muss an oder vor der Listdateiposition stehen. LINK gibt dieser Listdatei den Namen des Objektmoduls und fuegt die Standarddateierweiterung .MAP an.

Beispiel:

```
LINK datei/LI,,dat1 + bibn
```

Dieses Beispiel kopiert die Zeilennummern im Objektmodul datei.obj in die Listdatei datei.map.

3.6. Unterscheiden Gross- und Kleinbuchstaben

Syntax:

```
/NOIGNORECASE          Kurzschreibweise: /NOI
```

Der Schalter /NOI veranlasst den Binder Gross- und Kleinbuchstaben in Symbolen als unterschiedliche Buchstaben zu betrachten. Normalerweise betrachtet der Binder Gross- und Kleinbuchstaben als gleich.
Beispielsweise werden die Namen MARKE, marke und Marke durch den Binder standardmaessig als gleiche Symbole bewertet. Beim Anwenden von /NOI werden diese Symbole als drei verschiedene interpretiert.
Der Schalter /NOI wird meist in Verbindung mit Objektmoduln verwendet, die mit Compilern hoeherer Programmiersprachen erstellt wurden. Einige dieser Compiler interpretieren Gross- und Kleinbuchstaben als unterschiedliche Buchstaben und verlangen vom Binder eine gleichartige Interpretation. Fuer das Binden von Moduln, die mit MASM erstellt werden, hat der Schalter nur dann Bedeutung, wenn in MASM die Schalter /ML oder /MX verwendet und sie mit Moduln hoeherer Sprachen gebunden werden. Dann ist der Schalter /NOI notwendig.

Beispiel:

```
LINK datei1 + datei2/NOI,,, d1 + bibi
```

Dieses Kommando veranlasst den Binder, Gross- und Kleinbuchstaben als verschiedene Buchstaben in Symbolen zu betrachten.

3.7. Ignorieren von Standardbibliotheken

Syntax:

/NODEFAULTLIBRARYSEARCH Kurzschreibweise: /NOD

Dieser Schalter veranlasst den Binder, alle Bibliotheksnamen zu ignorieren, die er in einem Objektmodul findet. Ein Compiler hoeherer Programmiersprachen kann Bibliotheksnamen zu einem Objektmodul hinzufuegen, um zu sichern, dass ein standardmaessiges Binden mit dem Programm erfolgt.
Wird dieser Schalter benutzt, dann werden diese Standardbibliotheken uebergangen, und es koennen explizit die gewuenschten Bibliotheken in der LINK-Befehlszeile eingegeben werden.

Beispiel:

```
LINK  prog + datei/NOD,,, d1 + biba + bibc
```

Dieses Beispiel bindet die Objektmoduln prog.obj und datei.obj mit Routinen der Bibliotheken d1, biba und bibc. Alle Standardbibliotheken, die in prog.obj oder datei.obj vorkommen, werden ignoriert.

3.8. Setzen der Stackgroesse

Syntax:

/STACK: <zahl> Kurzschreibweise: /ST:<zahl>

Der /STACK-Schalter setzt den Programmstack auf die Zahl Bytes, die fuer <zahl> eingegeben werden. Der Binder berechnet die Programmstackgroesse, auf der Grundlage der einzelnen Stacksegmente, die in den Objektmoduln definiert werden, unterschiedlich. Wird der Schalter /ST verwendet, setzt der Binder die hier eingegebene Zahl von Bytes als Stackgroesse ein. Der errechnete Wert wird ignoriert. Fuer <zahl> kann jeder positive ganzzahlige Wert zwischen 1 und 65 535 eingegeben werden. Der Wert kann dezimal, oktal oder hexadezimal eingegeben werden. Oktalzahlen muessen mit einer Null beginnen, Hexadezimalzahlen dagegen mit einer fuehrenden Null und einem kleinen X (Bspl.0x1B).

Beispiele:

```
- LINK  datei/STACK:512,,;
```

Dieses Beispiel setzt den Stack auf 512 Byte.

```
- LINK  mod1 + modb,prog/ST:0xFF,d1,\bib\uprog;
```

Die Stackgroesse wird auf 255 (0FFH) Byte gesetzt.

```
- LINK  beg + datei/ST:030,,;
```

Die Stackgroesse wird auf 24 (30 oktal) Byte gesetzt.

3.9. Setzen der maximalen Anzahl von Leerzeichen

Syntax:

/CPARMAXALLOC:<zahl> Kurzschreibweise: /C:<zahl>

Der /C-Schalter setzt die maximale Anzahl von 16-Byte-Paragraphen, die vom Programm benoetigt werden, um es in den Speicher zu laden.
Die <zahl> wird durch das Betriebssystem genutzt, wenn die Zuweisung von Leerzeichen vorher notwendig ist, um das Programm laden zu koennen.
LINK setzt normalerweise das Maximum von Paragraphen auf 65 535. Das Betriebssystem erkennt diese Forderung nicht an und zeigt auf den groessten angrenzenden Speicherblock, den es finden kann. Die Zahl 65 535 Paragraphen bedeutet den ganzen adressierbaren Speicher.
Wenn der Schalter /C verwendet wird, fuegt das Betriebssystem nicht mehr Leerzeichen ein, als bei dem Schalter bei <zahl> eingegeben wurde. Das bedeutet, dass im Speicher mehr Platz fuer andere Programme entsteht. <zahl> kann ein ganzzahliger Wert im Bereich von 1 bis 65 535 sein. Sie kann dezimal, oktal oder hexadezimal eingegeben werden. Die Schreibweise ist im Gliederungspunkt 3.8. beschrieben.
Ist <zahl> kleiner als die Minimalzahl von Paragraphen, die im Programm benoetigt werden, ignoriert LINK die Eingabe und setzt den Maximalwert gleich dem Minimalwert, der benoetigt wird. Die Minimalzahl von Paragraphen, die durch das Programm benoetigt werden, ist nie kleiner als die Zahl der Paragraphen von Kode und Daten im Programm.
Der Schalter /C kann zum Binden von Dateien verwendet werden, bevor der Debugger verwendet wird. Damit kann in SYMDEB das Shell-Kommando verwendet werden (siehe SYMDEB).

Beispiele:

- **LINK datei/C:15,,;**

Dieses Beispiel setzt die Maximalzahl auf 15 Paragraphen.

- **LINK mod1+mod2+prog/C:0XFF,ab;**

Die Maximalzahl wird auf 255 Paragraphen (0FFH) gesetzt.

- **LINK progst+datei,/C:030,;**

Die Maximalzahl wird auf 24 (30 oktal) Paragraphen gesetzt.

3.10. Setzen einer hohen Startadresse

Syntax:

/HIGH Kurzschreibweise: /H

Der Schalter /HIGH setzt die Programmstartadresse an die hoechstmoegliche Adresse im freien Speicher. Wird der Schalter nicht verwendet, dann wird die Startadresse an die niedrigste Stelle im freien Speicher gesetzt.

Beispiel:

```
LINK startp+datei/H,,;
```

Dieses Beispiel setzt die Startadresse des Programmes startp.exe an die hoechstmoegliche Adresse im freien Speicheradressraum.

3.11. Zuweisung einer Datengruppe

Syntax:

/DSALLOCATE Kurzschreibweise: /D

Der Schalter /D veranlasst den Binder entgegengesetzt seiner normalen Arbeitsweise zu arbeiten, wenn Adressen bezueglich der Gruppe mit Namen DGROUP zugewiesen werden sollen. Normalerweise weisst LINK den Offset dem niedrigsten Byte einer Gruppe zu. Wird /D verwendet, weisst LINK den Offset FFFFH den hoechsten Byte in der Gruppe zu. Das Resultat ist, dass die Daten, die geladen werden sollen, so hoch wie moeglich in dem Speichersegment, das DGROUP enthaelt, eingeschrieben werden.
Werden sowohl /H als auch /D eingegeben, kann der untere Teil des DGROUP-Bereichs dynamisch von einem Anwenderprogramm durch dasselbe Segmentregister adressiert werden, wie die anderen Daten (ist nur bei Assemblerprogrammen sinnvoll).
Um die Gruppe verwenden zu koennen, muss ein Segmentregister auf die Startadresse von DGROUP gesetzt werden (mit Pseudoanweisung ASSUME in MASM).

Beispiel:

```
LINK pstart+datei/HIGH/DSALLOCATE,,,bibi+bibup
```

Dieses Beispiel veranlasst den Binder, das Programm so hoch wie moeglich in den Speicher zu laden und anschliessend die Offsets aller Dateibezeichnungen in DGROUP einzuordnen, so dass sie so hoch wie moeglich innerhalb der Gruppe geladen werden.

3.12. Entfernen von Gruppen aus einem Programm

Syntax:

/NOGROUPASSOCIATION Kurzschreibweise: /NOG

Der Schalter /NOG veranlasst den Binder Gruppenverbaende zu ignorieren, wenn Adressen zu Daten und Kode zugewiesen werden.
Dieser Schalter existiert hauptsaechlich fuer die Kompatibilitaet mit aelteren Versionen der Compiler. Er darf nur dann verwendet werden, wenn solche Objektmoduln gebunden werden sollen, die mit diesen Compilern erstellt oder mit Laufzeitbibliotheken, die diese alten Compiler enthalten, gearbeitet wird.

3.13. Setzen Ueberlagerungsinterrupt

Syntax:

/OVERLAYINTERRUPT: <zahl> Kurzschreibweise: /O:<zahl>

Der Schalter /O setzt die Interruptnummer der Ueberlagerungsroutine auf die eingegebene <zahl>. Der Schalter ueberschreibt die normale Ueberlagerungsinterruptnummer (03FH).Die <zahl> kann ein ganzzahliger Wert im Bereich von 0 bis 255 sein. Sie muss eine Dezimal-, Oktal- oder Hexadezimalzahl sein. Die Schreibweise ist im Gliederungspunkt 3.8. beschrieben.
MASM besitzt keinen Ueberlagerungsmanäger. Deshalb kann dieser Schalter nur verwendet werden, wenn Moduln von einem Sprachcompiler gebunden werden sollen, der Ueberlagerungsstrukturen schaffen kann.
Deshalb ist vorher die entsprechende Compilerdokumentation zu ueberpruefen, ob der Compiler diese Moeglichkeit besitzt, sonst ist dieser Schalter ungeeignet. Es duerfen auch keine Interruptnummern vergeben werden, die in Konflikt mit den DCPX-Standardinterrupts geraten koennen.

Beispiele:

- **LINK datei/O:255,,, bib1+bib2**

Die Ueberlagerungsnummer ist auf 255 gesetzt.

- **LINK mod1+mod2,prog/OVERLAY:0xff,list.map,bib1+bib2**

Die Ueberlagerungsnummer wird ebenfalls auf 255 (0FFH) gesetzt.

- **LINK startp+datei1/O:0377,, bib1+bib2**

Die Ueberlagerungsnummer wurde oktal (0377) eingegeben und entspricht dezimal 255.

3.14. Setzen der Maximalzahl von Segmenten

Syntax:

/SEGMENTS: <zahl> Kurzschreibweise: /SE: <zahl>

Der Segmentschalter veranlasst den Binder, nicht mehr als die eingegebene <zahl> von Segmenten im Bindelauf zu erlauben. Treten mehr Segmente auf, dann zeigt der Binder eine entsprechende Fehlermeldung an und unterbricht den Bindelauf.
Der Schalter wird verwendet, wenn das Standardlimit von 128 Segmenten ueberschrieben werden soll.
Falls der Schalter nicht angegeben wurde, hat der Binder nur soviel freien Speicherplatz zur Verfuegung, um maximal 128 Segmente verarbeiten zu koennen. Besitzt das zu bindende Programm jedoch mehr als 128 Segmente, wird der Schalter /SE unbedingt benoetigt, damit der Binder mehr als 128 Segmente verarbeiten kann.
Wird der Fehler

Segment limit set too high

angezeigt, ist im Schalter das Segmentlimit zu verringern. Die <zahl> kann jeden ganzzahligen Wert im Bereich von 1 bis 1024 annehmen. Sie muss eine Dezimal-, Oktal- oder Hexadezimalzahl sein. Die Schreibweise ist im Gliederungspunkt 3.8. beschrieben.

Beispiele:

- **LINK datei/SE:192,,;**

Dieses Beispiel setzt das Segmentlimit auf maximal 192 Segmente.

- **LINK modi+mod2,prog/SEGMENTS:0xff, bib1+bib2**

Das Segmentlimit wird auf maximal 255 (0FFH) Segmente heraufgesetzt.

3.15. Verwenden der DCP-Segmentordnung

Syntax:

/DOSSEG Kurzschreibweise: /DO

Der Schalter /DOSSEG veranlasst den Binder, alle Segmente in der ausfuehrbaren Datei gemaess der Segmentanordnungskonvention zu ordnen. Diese Konvention beinhaltet folgende Festlegungen:

1. Alle Segmente, die den Klassennamen 'CODE' besitzen, werden an den Beginn der ausfuehrbaren Datei platziert.
2. Alle anderen Segmente, das betrifft nicht die Gruppe, die den Namen DGROUP besitzt, werden sofort nach den 'CODE'-Segmenten abgelegt.
3. Alle Segmente, die in DGROUP zusammengefasst sind, werden an das Ende der Datei platziert.

Die normale Segmentordnung, wenn der Schalter /DO nicht verwendet wird, ist im Gliederungspunkt 4.3. erklaert.

Beispiel:

LINK stprog+datei/DOSSEG,,, bib1+bib2

Diese Befehlszeile veranlasst den Binder, eine ausfuehrbare Datei mit dem Namen stprog.exe zu generieren, deren Segmente in Uebereinstimmung mit der DCPX-Segmentanordnungskonvention gebracht wurden.

4. Arbeitsweise von LINK

LINK erstellt eine ausfuehrbare Datei durch das Verketten von Programmkode und Datensegmenten gemaess den Befehlen und Pseudooperationen in der Quelldatei. Diese verketteten Segmente bilden ein sogenanntes "ausfuehrbares Abbild", das direkt in den Speicher uebernommen wird, soll das Programm zur Abarbeitung gebracht werden. Folglich definieren Anordnung sowie Art und Weise, in der der Binder die Segmente in die ausfuehrbare Datei

kopiert, auch Anordnung und Art und Weise, in der die Segmente in den Speicher gebracht werden.
Dem Binder kann durch Uebergeben von Segmentattributen mit einer Segmentpseudooperation oder durch Verwenden der GROUP-Pseudooperation zum Bilden von Segmentgruppen, mitgeteilt werden, wie er die Segmente eines Programms binden soll.

Diese Pseudooperationen definieren Gruppenverbaende, Klassen und die Typen 'Align' und 'Combine', die die Rangfolge und die relativen Startadressen aller Segmente im Programm definieren.

4.1. Reihenfolge der Segmente

Der Binder verwendet einen Typ der Segmentreihenfolge, um die Startadressen fuer das Segment festzulegen. Dieser Typ kann **byte**, **word**, **para** oder **page** sein. Das bedeutet, dass die Startadressen an einer Byte-, Wort-, Paragraphen- oder Seitengrenze festgelegt wird (diese sind ein Vielfaches von: Byte=1, Word=2, Paragraph=16 und Seite=256 Bytegrenze).
Der Standardtyp ist "para", d.h. an jeder 16-Byte-Grenze kann ein neues Segment beginnen.
Findet der Binder ein Segment, prueft er den Typ der Segmentreihenfolge, bevor er das Segment in die ausfuehrbare Datei einordnet. Ist der Typ "word", "para" oder "page", prueft der Binder, ob das letzte uebernommene Byte an solch einer Grenze steht. Ist das nicht der Fall, dann fuellt der Binder die Bytes bis zur entsprechenden Grenze mit Ø auf.

4.2. Rahmennummer

Der Binder erstellt eine Startadresse fuer jedes Segment in einem Programm. Die Startadresse basiert auf dem Segmentreihenfolgetyp und der Groesse der Segmente, die bereits in die ausfuehrbare Datei kopiert wurden. Die Adresse besteht aus einem Offset und aus einer speziellen Rahmennummer. Diese Rahmennummer spezifiert die Adresse des ersten Paragraphen im Speicher, der ein oder mehrere Bytes des Segmentes enthaelt. Die Rahmennummer ist immer ein Mehrfaches von 16 (eine Paragraphenadresse). Der Offset ist die Zahl der Bytes vom Start des Paragraphen zum ersten Byte in dem Segment. Fuer Byte- und Worttypen kann der Offset auch nicht Null sein. Er ist immer Null fuer para- und page-Typen. Die Rahmennummer eines Segmentes kann von einer LINK-Datei erhalten werden. Es sind die ersten 5 Hexastellen der Startadresse, die fuer das Segment spezifiert werden.

4.3. Anordnung der Segmente

LINK kopiert die Segmente in der gleichen Reihenfolge, wie sie in den Objektmoduln stehen, in die ausfuehrbare Datei. Diese Anordnung wird im Programm aufrechterhalten, wenn der Binder auf zwei oder mehrere Segmente trifft, die den gleichen Klassennamen besitzen. Segmente, die identische Klassennamen in Bezug auf den gleichen Klassentyp haben, werden in die ausfuehrbare Datei als

benachbarte Bloecke kopiert.

4.4. Kombinierte Segmente

Die entsprechenden Typen, die entscheiden, ob zwei oder mehrere Segmente mit gleichen Segmentnamen in einem einzelnen grossen Segment abgelegt (kombiniert) werden, sind: **public, stack, common, memory, at** und **private.**
Hat ein Segment den Typ "public", dann kombiniert der Binder automatisch dieses Segment mit allen anderen Segmenten, die den gleichen Segmentnamen haben und der gleichen Klasse angehoeren. Kombiniert der Binder Segmente, sichert er, dass die Segmente aufeinanderfolgen und dass auf alle Adressen in den Segmenten ueber einen Offset von der Anfangsadresse zugegriffen werden kann. Das Resultat ist das Gleiche, als waeren die Segmente als Ganzes in der Quelldatei definiert worden. Der Binder bewahrt somit jeden individuellen Segmentreihenfolgetyp.
Die Segmente werden lediglich zu einem grossen Segment zusammengefasst, der Kode und die Daten in den Segmenten behalten ihren originalen Reihenfolgetyp. Ueberschreitet ein solches kombiniertes Segment 64 KByte, dann zeigt LINK eine Fehlermeldung an.
Hat ein Segment den Typ "stack", fuehrt der Binder die gleiche Operation aus, wie fuer Segmente mit "public". Der einzige Unterschied ist, dass die Stacksegmente den Binder veranlassen, einen Stackpointeranfangswert in die ausfuehrbare Datei zu kopieren. Dieser Stackpointerwert ist der Offset bis zum Ende des ersten Stacksegments. Beim Verwenden des Types "stack" fuer Stacksegmente sind keine Befehle zum Laden des Segments in das SS-Register notwendig.
Hat ein Segment den Typ "common", kombiniert der Binder dieses Segment automatisch mit allen anderen Segmenten, die den gleichen Namen und die gleiche Klasse haben. Werden LINK "Common"-Segmente kombiniert, setzt er die Startadresse eines jeden Segmentes auf die gleiche Adresse und erzeugt damit eine Serie von sich ueberlappenden Segmenten. Das Ergebnis ist ein Segment, das nicht groesser als das groesste Segment der kombinierten Segmente ist.
Die Segmente mit dem Typ "memory" werden vom Binder wie Segmente des Types "public" behandelt.
Ein Segment hat den Typ "private", wenn kein anderer Typ in der Quelldatei definiert wurde.

4.5. Gruppen

Gruppen erlauben nur das Adressieren benachbarter Segmente, gleicher Klasse, relativ zur Anfangsadresse des ersten Segmentes. Erkennt der Binder eine Gruppe, ordnet er alle Speicherbezugnahmen zu Ausdruecken in der Gruppe so, dass sie relativ zur Anfangsadresse stehen. Segmente in einer Gruppe brauchen nicht aufeinanderfolgen und nicht die gleiche Klasse und Typ zu besitzen. Die einzige Bedingung ist, dass alle Segmente einer Gruppe zusammen nicht mehr als 64 KByte Speicherplatz belegen duerfen.
Gruppen beeinflussen nicht die Reihenfolge, in der die Segmente geladen werden. Werden jedoch Klassennamen verwendet und die

Objektmoduln in der richtigen Reihenfolge eingegeben, ist das keine Garantie dafuer, dass die Segmente aufeinanderfolgend geladen werden. Daraus ergibt sich, dass der Binder auch Segmente, die nicht zur Gruppe gehoeren, in die gleichen 64 KByte Speicherplatz einordnet.
Er ueberprueft nicht, ob alle Segmente einen 64 KByte Speicherplatz fuellen, er bringt lediglich den Fehler "fixup-overflow", wenn diese Forderung nicht erfuellt wird.
Die Erklaerung ueber Gruppen erfolgt in der Beschreibung MASM.

4.6. Finden nicht aufgeloester Bezugsnahmen

Sobald die Startadressen eines jeden Segments ermittelt und alle Segmentkombinationen und Gruppen eingeordnet sind, kann der Binder alle nicht aufgeloesten Bezugsnahmen zu Marken und Variablen finden. Um diese Bezugsnahmen festzustellen, berechnet LINK ein entsprechendes Offset und eine Segmentadresse und ersetzt die vorlaeufigen Werte, die durch den Assembler generiert wurden, durch die neuen Werte.
LINK berechnet die Bezugsnahmen durch 4 verschiedene Moeglichkeiten:

- Kurzdistanz (short)
- im Segment selbst relativ (near self-relative)
- im Segment relativ (near segment-relative)
- Langdistanz, ueber die Segmentgrenze hinaus (long)

Die Groesse des errechneten Wertes, ist abhaengig vom Typ der Bezugsnahme. Findet der Binder einen Fehler in der Groesse einer Bezugsnahme, zeigt er den Fehler "fixup-overflow" an.
Dieser Fehler kann auftreten, wenn in einem Programm mit einem 16-Bit-Offset versucht wird, einen Befehl in einem anderen Segment (ueber die Segmentgrenze hinaus) zu erreichen. Passen alle Segmente einer Gruppe nicht in einen 64 KByte Speicherblock, wird der Fehler ebenfalls angezeigt.
Eine Kurzdistanzbezugnahme kommt in Sprungbefehlen und Schleifenbefehlen vor. Der Zielbefehl darf nicht mehr als 128 Byte von der Bezugnahme entfernt sein. Der Binder erzeugt fuer diese Bezugnahme eine vorzeichenbehaftete 8-Bit-Zahl. Es wird eine Fehlermeldung angezeigt, wenn sich der Zielbefehl in einem anderen Segment oder in einer anderen Gruppe befindet, oder wenn er mehr als 128 Byte (Distanz) entfernt ist.
Eine "im Segment selbst relative Bezugnahme" kommt bei Befehlen vor, deren Datenzugriff relativ zum gleichen Segment oder der gleichen Gruppe durchgefuehrt wird. Der Binder erzeugt einen 16-Bit Offset fuer diese Bezugnahme.
Befinden sich die entsprechenden Daten nicht im gleichen Segment oder der gleichen Gruppe, erfolgt durch LINK eine Fehleranzeige.
Eine "im Segment relative Bezugnahme" kommt bei Befehlen vor, die auf Daten in einem spezifierten Segment, einer Gruppe oder relativ zu einem spezifierten Segmentregister zugreifen. Der Binder erzeugt einen 16-Bit-Offset fuer diese Bezugnahme. Es wird eine Fehlermeldung angezeigt, wenn der Zieloffset groesser als 64 KByte oder kleiner als Null oder das Ziel nicht adressierbar ist.
Eine "Langdistanzbezugnahme" kommt bei CALL- und JMP-Befehlen

vor, die auf einen Befehl in einem anderen Segment oder einer anderen Gruppe zugreifen. Der Binder erzeugt eine 16-Bit-Anfangsadresse (fuer das entsprechende Segment) und einen 16-Bit-Offset fuer die Bezugnahme. LINK meldet einen Fehler, wenn der Offset groesser als 64 KByte oder kleiner als Null bzw. der Zielbefehl nicht adressierbar ist.

5. Fehlermeldungen von LINK

In diesem Kapitel sind alle Fehlermeldungen aufgefuehrt, die beim Binden von Objektmoduln auftreten koennen. Die Meldungen sind alphabetisch geordnet.

- Ambiguos switch error: "schalter"

  Es wurde keine oder nur eine unvollstaendige Bezeichnung eines Schalters nach dem Schraegstrich eingegeben.

  Beispiel: LINK /N prog;

  Der Fehler wird angezeigt, weil der Binder nicht weiss, welcher der drei moeglichen Schalter, die mit "N" beginnen, verwendet werden soll.

- Array element size mismatch

  Dieser Fehler kann nur auftreten, wenn die Objektmoduln mit Compilern erzeugt wurden, die gemeinsame Felder enthalten. Der Fehler kann bei Objektmoduln, die mit MASM erstellt wurden, nicht auftreten.

  Beispiel: Das Feld wurde einmal als Zeichenkettenfeld und ein .weiteres Mal als numerisches Feld deklariert.

- Attempt to put segment <name> in more than one group in file <dateiname>

  Ein Segment wurde als Bestandteil von zwei verschiedenen Gruppen definiert. Das Quellprogramm muss korrigiert und ein neuer Objektmodul erstellt werden.

- Bad value for CparMaxAlloc

  Die beim Schalter /CPARMAXALLOC spezifierte Zahl ist nicht im Bereich von 1 bis 65 535.

- Cannot find library: dateiname.lib Enter new file spec:

  Der Binder kann den eingegebenen Bibliotheksnamen (dateiname.lib) nicht finden. Der Nutzer wird aufgefordert, eine neue Dateibezeichnung fuer die Bibliothek einzugeben.

- Cannot open list file

  Die Diskette oder das Stammverzeichnis ist voll. Es muessen

Dateien geloescht oder eine neue Diskette verwendet werden.

- Cannot open response file

Der Binder kann die Antwortdatei, die durch den Nutzer spezifiert wurde, nicht finden. Wahrscheinlich liegt hier meist ein Eingabefehler vor.

- Cannot nest response files

Eine Antwortdatei in einer Antwortdatei darf nicht verwenden werden (keine Verschachtelung!).

- Cannot open run file

Die Diskette oder das Stammverzeichnis ist voll. Es muessen Dateien geloescht oder eine neue Diskette eingelegt werden.

- Cannot open temporary file

Die Diskette oder das Stammverzeichnis ist voll. Es muessen Dateien geloescht oder eine neue Diskette eingelegt werden.

- Cannot reopen list file

Die Originaldiskette wurde trotz Aufforderung nicht wieder eingelegt. Der Bindelauf ist von vorn zu beginnen.

- Common area larger than 65 535 bytes

Das Nutzerprogramm hat mehr als 64 KByte gemeinsame Variablen. Dieser Fehler kann bei Objektmoduln, die durch MASM generiert wurden, nicht auftreten. Er ist nur moeglich, wenn zum Generieren der Objektmoduln Compiler hoeherer Programmiersprachen verwendet wurden, die gemeinsame Variablen erlauben.

- Data record too large

Der LEDATA-Satz (in einem Objektmodul) enthaelt mehr als 1024 Byte Daten. Dies ist ein Uebersetzungsfehler. Es ist zu beachten, dass der Uebersetzer (Compiler oder Assembler) den nicht korrekten Objektmodul und damit den Fehler produziert hat. LEDATA ist ein DCP-Ausdruck.

- Dup record too large

Der LIDATA-Satz (in einem Objektmodul) enthaelt mehr als 512 Bytes Daten. Das geschieht, wenn ein Assemblermodul eine Strukturdefinition enthaelt, die sehr komplex ist und eine Serie von verschachtelten DUP-Operatoren enthaelt.

Beispiel: FELD DB 10DUP(11DUP(12DUP(13DUP(...))))

Die Beseitigung ist nur durch Entflechten und Vereinfachen in der Quelldatei moeglich. Anschliessend muss neu assembliert werden. LIDATA ist ein DCP-Ausdruck.

- Filename is not a valid library

  Die spezifierte Datei als Bibliotheksdatei ist falsch. Der Binder bricht ab.

- Fixup overflow near **number** in Segment **name** in **filename** offset **number**

  Dieser Fehler tritt in folgenden Faellen auf:

  1. Eine Gruppe ist groesser als 64 KByte.
  2. Das Nutzerprogramm enthaelt einen kurzen Sprung oder UP-Aufruf in ein anderes Segment.
  3. Der Nutzer hat in einem Objektmodul einen Datenausdruck, dessen Name mit einem Unterprogrammnamen in Konflikt geraet, der zum Binden mit genutzt wird.
  4. Es wurde eine EXTRN-Pseudooperation innerhalb der Grenzen eines Segmentes spezifiert.

  Beispiel:

```
CODE    SEGMENT     PUBLIC  'CODE'
        EXTRN       HAUPT: FAR
BEG     PROC        FAR
        CALL        HAUPT
        RET
BEG     ENDP
CODE    ENDS
```

  Richtig muesste dieses Programmstueck lauten:

```
        EXTRN       HAUPT: FAR
CODE    SEGMENT     PUBLIC  'CODE'
BEG     PROC        FAR
        ,
        ,
        ,
        RET
BEG     ENDP
CODE    ENDS
```

  Der Fehler muss in der Quelldatei korrigiert werden. Danach ist neu zu uebersetzen.

- Incorrect DCPX version, use DCP 2.0 or later

  Der Binder laeuft nur ab Betriebssystemversion 2.0 und hoeher. Es muss ein geeignetes Betriebssystem geladen und der Lauf wiederholt werden.

- Insufficient stack space

  Es ist nicht mehr genug Speicherplatz vorhanden, um den Bindelauf durchfuehren zu koennen.

- Interrupt number exceeds 255

  Es wurde eine Zahl groesser 255 als Wert beim Schalter /OVERLAYINTERRUPT eingegeben. Der Lauf muss mit einem Wert im Bereich von 0 bis 255 wiederholt werden.

- Invalid numeric switch specification

  Es wurde ein unkorrekter Wert fuer einen der Schalter des Binders eingegeben.

  Beispiel:

  Eingabe eines Zeichens, der Schalter erwartet einen numerischen Wert.

- Invalid object module

  Einer der Objektmoduln in der Liste ist nicht korrekt.

- NEAR/HUGE conflict

  Widersprueche bei gemeinsamen Variablen bei "nahen" und "grossen" Definitionen sind vorhanden. Dieser Fehler kann bei Objektmoduln, die mit MASM generiert wurden, nicht auftreten. Er kann nur bei Moduln auftreten, die mit Compilern hoeherer Programmiersprachen erstellt wurden, die gemeinsame Variablen erlauben.

- Nested left parentheses

  Es tritt ein Eingabefehler waehrend des Spezifizierens einer Ueberlagerung in der Befehlszeile auf. Deshalb ist es guenstig, bei diesem Fehler im entsprechenden Compilerhandbuch nachzulesen, wie Ueberlagerungen fuer den Binder erstellt werden muessen. Beim Binden von Objektmoduln, die mit MASM generiert wurden, kann dieser Fehler nicht auftreten.

- No object module specified

  In der Befehlszeile ist mindestens ein Objektmodul einzugeben.

- Out of space on list file

  Die Diskette, auf der die Listdatei geschrieben werden soll, ist voll. Es muss Platz auf der Diskette geschaffen (loeschen von Dateien) oder eine neue Diskette eingelegt werden. Anschliessend wird der Lauf wiederholt.

- Out of space on run file

  Die Diskette, auf der die ausfuehrbare Datei geschrieben werden soll, ist voll. Die Fehlerbeseitigung erfolgt wie beim vorherigen Fehler.

- Out of space on scratch file

  Dei Diskette im Standardlaufwerk ist voll. Fehlerbeseitigung erfolgt analog dem Fehler "Out of space on list file".

- Overlay manager symbol already defined: name

  Es wurde ein Symbolname definiert, der sich mit einem Spezialueberlagerungsmanagernamen widerspricht. Der unkorrekte Name ist auszutauschen und der Bindelauf zu wiederholen.
  MASM besitzt keinen Ueberlagerungsmanager. Dieser Fehler kann nur auftreten, wenn eine Bibliothek, die mit Compilern hoeherer Programmiersprachen (die Ueberlagerungen erlauben) erstellt wurde, mit eingebunden wird.

- Relocation table overflow

  Im Programm sind mehr als 32 768 UP-Aufrufe und Spruenge in andere Segmente (long call, long jump) oder andere Zeiger in solche Segmente enthalten. Das Programm muss neu gestaltet werden und solche "langen Bezugnahmen", wenn moeglich, durch Kurze zu ersetzen. Dann kann ein neuer Objektmodul erstellt werden.
  Es muss weiterhin beachtet werden, dass PASCAL- und FORTRAN-Nutzer zuerst den Testschalter einschalten muessten.

- Segment limit set too high

  Das Limit der maximalen Anzahl der Segmente im Programm wurde beim Schalter /SEGMENTS zu hoch gesetzt (ueber 1024).

- Segment limit too high

  Fuer den Binder ist nicht mehr genug Speicherplatz vorhanden, um die Tabellen zu durchlaufen, die die Anzahl der Segmente fordert (Standard ist 128 oder es wurde mit /SEGMENTS ein neuer Wert eingegeben). Es muss mit dem Schalter /SEGMENTS eine kleinere Zahl angegeben oder freier Speicherplatz durch das Loeschen von residenten Programmen oder Shells geschaffen werden. Anschliessend ist der Bindelauf zu wiederholen.

- Segment size exceeds 64 K

  Der Nutzer hat ein kleines Modellprogramm mit mehr als 64 KByte Kode oder ein mittleres Modellprogramm mit mehr als 64 KByte Daten. Es muss versucht werden ein mittleres oder grosses Modellprogramm zu compilieren und zu binden.

- Stack size exceeds 65 536 byte

  Die Stackgroesse, durch den Schalter /STACK festgelegt, ist groesser als 65 536 Byte. Es muss mit Stack eine erlaubte Groesse eingegeben werden.

- Symbol table overflow

  Symboltabellenueberlauf
  Das Nutzerprogramm hat mehr als 256 KByte symbolische Informationen (Eintrittspunkte, externe Bezugsnahmen, Gruppen, Klassen, Dateien usw.). Zur Fehlerbeseitigung muessen Moduln oder Segmente kombiniert oder so viele Symbole wie moeglich, eleminiert werden. Anschliessend sind neue Objektmoduln zu erstellen und der Bindelauf kann wiederholt werden.

- Terminated by user

  Der Bediener hat CTRL-C betaetigt, was zum Abbruch des Bindelaufes fuehrt.

- Too many external symbols in one module

  Es wurden mehr als 1024 externe Symbole in einem Modul spezifiert.

- Too many group-, segment- and class-names in one module

  Das Programm enthaelt in einem Modul zu viele Gruppen-, Segment- und Klassennamen. Die Anzahl dieser Namen ist zu reduzieren und anschliessend kann der Bindelauf wiederholt werden.

- Too many groups

  Es wurden mehr als 9 Gruppen im Programm definiert. Die Anzahl der Gruppen muss reduziert werden.

- Too many GRPDEFS in one module

  LINK stellt mehr als 9 Gruppendefinitionen in einem einezelnen Modul fest. Zur Fehlerbeseitigung ist die Gruppenanzahl zu verringern oder der Modul zu teilen.

- Too many libraries

  Es wurde versucht mehr als 16 Bibliotheken zu binden. Zur Fehlerbeseitigung muessen die Bibliotheken kombiniert oder Moduln veraendert werden, damit weniger Bibliotheken notwendig sind.

- Too many overlays

  Es wurden mehr als 63 Ueberlagerungen definiert. Die Anzahl der Ueberlagerungen muss reduziert werden. Bei mit MASM erstellten Moduln kann dieser Fehler nicht auftreten.

- Too many segments

  Das Programm hat mehr Segmente, als erlaubt sind (standardmaessig 128). Der Bindelauf ist mit dem Schalter /SEGMENTS, der eine hoehere Anzahl von Segmenten gestattet, zu wiederholen.

- Too many segments in one module

  Ein Objektmodul hat mehr als 255 Segmente. Zur Fehlerbeseitigung muss der Modul geteilt oder Segmente kombiniert werden.

- Too many TYPDEFs

  Ein Modul enthaelt zu viele TYPDEF-Saetze. Diese Saetze beschreiben gemeinsame Variablen. Dieser Fehler tritt bei Moduln, die mit MASM generiert wurden, nicht auf. Er kann nur bei Programmen auftreten, die durch Compiler hoeherer Programmiersprachen, die gemeinsame Variablen erlauben, generiert wurden.

- Unexpected end-of-file on scratch file

  Die Diskette mit der Datei VM.TMP wurde aus dem Laufwerk entnommen (siehe Punkt 2.6).

- Unmatched left parenthesis

  Der Inhalt einer Ueberlagerung wurde in der Befehlszeile spezifiert. Es muss im jeweiligen Compilerhandbuch nachgelesen werden, wie solche Ueberlagerungen fuer LINK spezifiert werden muessen.
  MASM enthaelt keinen Ueberlagerungsmanager, so dass dieses Problem nur beim Einbinden von Bibliotheken hoeherer Programmiersprachen auftreten kann.

- Unmatched right parenthesis

  Fehlerart und Auftreten, wie beim Fehler "Unmatched left parenthesis".

- Unrecognized switch error: "schalter"

  Es wurden nach dem Schraegstrich Buchstaben eingegeben, diese sind keine gueltigen Schalter.

  Beispiel: LINK /XYZ main;

- Unresolved externals

  Ein Symbol wurde in einem Modul als extern deklariert, jedoch in dem definierten Modul nicht als "PUBLIC".
  Ein solches Symbol muss mit der Pseudooperation "PUBLIC" in dem Modul deklariert werden, wo es definiert wird, bevor es als externes Symbol in anderen Moduln mit der Pseudooperation "EXTRN" verwendet werden kann.

- VM.TMP is an illegal file name and has been ignored

  Der nichterlaubte Modulname VM.TMP wurde vergeben. Dieser Modul ist umzubenennen und der Bindelauf kann wiederholt werden.

- Warning: no stack segment

  Das Programm enthaelt kein Stacksegment, das durch den Typ "stack" definiert wurde. Diese Meldung kann jedoch ignoriert werden, wenn ein Stack ohne diesen "Stack"-Typ definiert wurde oder eine besondere Programmversion vorliegt.

- Warning: too many public symbols

  Der Schalter /MAP wurde verwendet, um eine sortierte Liste aller Public-Symbole in der Listdatei aufzustellen. Das Programm enthaelt aber zu viele Symbole fuer die Sortierung. Der Binder erzeugt in diesem Falle eine unsortierte Liste dieser Symbole.

VI. DEBUGGER SYMDEB

1. Einfuehrung

SYMDEB ist ein Dienstprogramm zum Testen von ausfuehrbaren Dateien. Das Anzeigen und Ausfuehren von Instruktionen, das Setzen von Unterbrechungspunkten und das Anzeigen und Aendern von Werten im Speicher sind einige Funktionen von SYMDEB.
SYMDEB kann auf Programmpunkte durch Adressen oder globale Symbole Bezug nehmen.

Bei der Syntaxbeschreibung gilt folgende Notation:

BE	Grossbuchstabe fuer Schluesselwoerter
<bereich>	Kleinbuchstaben in spitze Klammern eingeschlossen fuer zu ersetzende Begriffe
[ ]	wahlweise Angabe
|	alternative Angabe

2. Symbolisches Testen

Um die Moeglichkeit des symbolischen Testens von SYMDEB nutzen zu koennen, muss eine spezielle Symboldatei erzeugt werden. Dies geschieht in folgenden Schritten:

a) Alle Symbole, die in SYMDEB benutzt werden sollen, muessen als PUBLIC deklariert werden(Segment- und Gruppennamen gelten automatisch als PUBLIC).

b) Das Quellprogramm wird mit dem Assembler MASM uebersetzt. Zweckmaessigerweise erstellt man zur Testerleichterung eine Uebersetzungsliste, z.B.:

```
MASM test ,,;
```

c) Durch den Bindelauf wird eine ausfuehrbare Version des Programmes erzeugt. Es muessen eine MAP-Datei gebildet und die IMAP-Option in der Link-Kommandozeile benutzt werden, z.B.:

```
LINK test ,, /MAP;
```

d) Das Programm MAPSYM liefert eine von SYMDEB verarbeitbare Symboldatei:

```
MAPSYM test
```

3. MAPSYM

Symboldateien, die Daten fuer den symbolischen Test enthalten, werden mit dem Programm MAPSYM erstellt. Das Programm wandelt den Inhalt der MAP-Symboldatei in die von SYMDEB benoetigte Form um. Die erzeugte Symboldatei kann bis zu 10 000 Symbole je Segment enthalten und so viele Segmente, wie es der verfuegbare

Speicherplatz der Anlage erlaubt. Das entsprechende Quellprogramm darf jedoch nicht mehr als 10 000 Quellzeilen besitzen.

Die MAPSYM-Kommandozeile hat die Form:

MAPSYM [/L|-L]<mapdateiname>

Der <mapdateiname> ist der Dateiname (wahlweise auch Pfadname) einer Symboldatei (.MAP), die durch den Binder erzeugt wurde. Ist keine Erweiterung angegeben, wird .MAP angenommen. Es ist zu beachten, dass beim Binden die /MAP-Option angenommen werden muss.

Die /L-Option bei MAPSYM bewirkt die Anzeige der Gruppennamen, der Programm-Startadresse, der Anzahl der Segmente und der Symbole je Segment auf dem Bildschirm. Die /L-Option kann auch als -L, l oder -l angegeben werden.

Beispiel:

MAPSYM /L datei

MAPSYM wandelt die in datei.map enthaltenen Daten um und erzeugt die Datei datei.sym, dabei erfolgt eine Anzeige von Informationen auf dem Bildschirm.

Beachte:

Die Symboldatei (.SYM) wird stets im aktuellen Laufwerk und aktuellen Verzeichnis erzeugt. Man kann in der Kommandozeile weder ein Ziel-Laufwerk noch ein Verzeichnis angeben.

Soll zum Beispiel test.sym auf Laufwerk B erzeugt werden, MAPSYM steht in Laufwerk A zur Verfuegung und test.map in Laufwerk B, so muessen

```
A>B:
B>A:MAPSYM test
```

eingegeben werden.

---

4. Start von SYMDEB

Die SYMDEB-Kommandozeile hat die allgemeine Form:

SYMDEB [<option>][<symboldateien>][<ausfuehrbares programm>]
[<argumente>]

Als <option> koennen eine oder mehrere angegeben werden, <symboldateien> sind die Namen von Symboldateien und <ausfuehrbares Programm> der Name einer COM- oder EXE-Datei. Die <argumente> sind Parameter des ausfuehrbaren Programms. Nach dem

Start erfolgt eine Programmanzeige, und das Promptzeichen (-) kennzeichnet die Bereitschaft zur Kommandoeingabe.

4.1. Start mit nur einer ausfuehrbaren Datei

Man kann eine ausfuehrbare Datei (.COM, .EXE) durch Angabe ihres Namens in der SYMDEB-Kommandozeile laden.

Nach dem Laden einer ausfuehrbaren Datei bildet SYMDEB einen 256-Byte-Kennsatz im niedrigsten verfuegbaren Speichersegment und kopiert den Dateiinhalt unmittelbar nach dem Kennsatz in den Speicher. Die Dateilaenge (in Bytes) wird im BX:CX-Registerpaar (in CX niederwertiger Teil) abgelegt und entsprechend dem Dateiinhalt Segment- und andere Register initialisiert.

Beachte:

Ist die Datei eine EXE-Datei, so wird der in der Datei enthaltene Kennsatz von SYMDEB zwar ausgewertet, aber nicht mit in den Speicher uebernommen. Daher stimmt die in BX:CX angegebene Laenge nicht mit der Gesamt-Dateilaenge ueberein.

4.2. Start fuer symbolisches Testen

Beim Entwickeln und Testen von Programmen ist es meist bequemer, auf Daten und Instruktionen durch Namen als durch Adressen zuzugreifen. Das erfordert aber die Angabe einer oder mehrerer Symboldateien zusammen mit einer ausfuehrbaren Datei in der Kommandozeile.

Die Angabe nicht nur einer, sondern mehrerer Symboldateien ist typisch fuer Programme, die aus mehreren ausfuehrbaren Dateien bestehen (z.B. Programme, die andere Programme aufrufen oder Geraetetreiber verwenden). Die Namen aller Symboldateien muessen vor dem Namen der ausfuehrbaren Datei angegeben werden.

Werden mehrere Symboldateien geladen, so wird nur eine als Symboltabelle eroeffnet, besitzt eine der Symboldateien den gleichen Namen wie die ausfuehrbare Datei, dann diese. Sonst wird die erste in der Kommandozeile angegebene Symboldatei eroeffnet.

Waehrend eines SYMDEB-Laufes kann mit einem Kommando (XD) eine andere Symboldatei (Symboltabelle) eroeffnet werden, die vorher eroeffnete Symboldatei wird geschlossen, weil nur jeweils eine aktiviert werden kann.

Man braucht nicht unbedingt eine ausfuehrbare Datei zusammen mit Symboldateien zu laden, z.B. wenn das zu lesende Programm bereits im Speicher resident ist oder spaeter waehrend des SYMDEB-Laufes mit den Kommandos N und L geladen werden soll.

Beachte:

Um eine falsche Adresszuordnung zu vermeiden, sollten Symboldateien vor dem Laden nicht umbenannt werden.

Beispiel:

```
SYMDEB test1.sym test.sym test.exe
```

SYMDEB laedt die Symboldateien test1.sym und test.sym, erzeugt einen Kennsatz im Speicher und laedt danach test.exe. Die Symboldatei (Symboltabelle) test.sym wird anstelle test1.sym eroeffnet, da sie den gleichen Namen wie die ausfuehrbare Datei hat.

4.3. Programmargumente

Man kann durch Angabe unmittelbar nach dem Programmnamen in der Kommandozeile das Programm mit Argumenten versehen. Diese Argumente werden in der eingegebenen Form in den Programm-Kennsatz uebernommen.

Beispiel:

```
SYMDEB test.exe param1 param2 param3 param4

- D 5d 9f
2689:0050                                               50 41 52
      PAR
2689:0060 41 4D 31 20 20 20 20 20 00 00 00 00 00 50 41 52
AM1    PAR
2689:0070 41 4D 32 20 20 20 20 20 00 00 00 00 00 00 00 00
AM2
2689:0080 1C 20 70 61 72 61 6D 31 20 70 61 72 61 6D 32 20
param1 param2
2689:0090 70 61 72 61 6D 33 20 70 61 72 61 6D 34 0D 00 00
param3 param4
-
```

Im Beispiel wurde das D-Kommando benutzt, um den Inhalt des Programm-Kennsatzes nach dem Laden zu zeigen. Die ersten beiden Argumente werden als Dateinamen in die Standard-Dateisteuerbloecke eingetragen. Diese Bloecke beginnen ab Byte 5Dh und 6Dh des Programm-Kennsatzes. Die Laenge der Argumenten-Liste enthaelt das Byte an 80h, eine genaue Kopie der Liste beginnt ab Byte 81h des Kennsatzes (weitere Details ueber den Programm-Kennsatz siehe N-Kommando).

4.4. Start ohne Datei

Beim Start von SYMDEB ohne Dateinamen in der Kommandozeile wird ebenfalls ein Programm-Kennsatz im Speicher erzeugt. Man kann dann entweder ein Programm mit dem A- oder E-Kommando erzeugen oder mit dem Kommando N und L eine beliebige Datei nachladen.
Beim Start von SYMDEB ohne Datei werden die Segment-Register auf die Anfangsadresse des freien Speicherbereiches gesetzt, der Befehlszaehler (IP) enthaelt den Wert 0100h, alle Flags werden geloescht und die restlichen Register enthalten den Wert 0.

Beispiel:

```
SYMDEB

- R
AX=0000  BX=0000  CX=0000 DX=0000 SP=FFEE  BP=0000  SI=0000
DI=0000
DS=26B2 ES=26B2 SS=26B2 CS=26B2 IP=0100 NV UP DI PL NZ NA
PO NC
```

Im Beispiel wurde nach dem Laden das R-Kommando eingegeben, um die Register-Initialisierung anzuzeigen.

5. SYMDEB-Optionen

Folgende Optionen koennen in der SYMDEB-Kommandozeile angegeben werden:

Option	Wirkung
/K	Unterbrechungstaste
/S	Bildschirm-Anzeigewechsel
/"<kommandos>"	Startkommandos

Als Kennzeichen koennen zusaetzlich Schraegstrich (/) oder Minuszeichen eingegeben werden, die Buchstaben koennen Gross- oder Kleinbuchstaben sein.

Beachte:

Dateien, deren Name ein Minuszeichen enthaelt, muessen vor dem Benutzen durch SYMDEB umbenannt werden, weil das Minuszeichen als Kennzeichen fuer eine Option interpretiert wird.

5.1. Interaktive Unterbrechungstaste

Syntax:

```
/K | -K
```

Die /K-Operation bewirkt, dass die SCROLL-LOCK-Taste als interaktive Unterbrechungstaste genutzt werden kann. Damit kann durch ihre Betaetigung die Programmausfuehrung unterbrochen werden, beispielsweise bei Endlos-Schleifen, die mit dem Kommando G gestartet werden.

Wartet allerdings das Programm auf eine Eingabe, kann durch CONTROL-C stets (auch ohne /K-Option) die Programmausfuehrung unterbrochen werden.

Beispiel:

```
SYMDEB /K test.sym test.exe
```

5.2. Bildschirm-Anzeigewechsel

Syntax:

```
/S | -S
```

Die /S-Option erlaubt das Umschalten zwischen der Anzeige von SYMDEB und der Anzeige des zu testenden Programms. Diese Moeglichkeit ist besonders nuetzlich bei Grafikprogrammen und solchen, die viele Daten auf dem Bildschirm anzeigen.

Jedoch belegt die /S-Option einen Speicherbereich von 32 KByte. Sie kann nicht bei einem Grafikmodus verwendet werden, der mehr als 32 KByte benutzt.

Beispiel:

```
SYMDEB /S test.sym test.exe
```

5.3. Startkommandos

Syntax:

```
/"<kommados>"|-"<kommandos>"
```

Die Startkommando-Option veranlasst SYMDEB zur automatischen Ausfuehrung der zwischen den Anfuehrungszeichen stehenden Kommandos unmittelbar nach dem Start. Die Moeglichkeit kann zum Beispiel beim Start von SYMDEB von einer Stapelverarbeitungs-Datei (.BAT) genutzt werden. Ein Semikolon (;) trennt die einzelnen Kommandos in der Liste.

Beispiel:

```
SYMDEB /"d80;U;r" test.exe
```

SYMDEB laedt test.exe, zeigt den Speicherinhalt ab 80h an, reassembliert die ersten fuenf Instruktionen und zeigt den Inhalt der Register an.

6. Kommando-Parameter

SYMDEB-Kommandos haben folgende allgemeine Form:

<kommandoname> <parameter>

<kommandoname> besteht aus einem oder zwei Buchstaben und <parameter> sind Zahlen, Symbole oder Ausdruecke, die Werte oder Adressen darstellen. Die Eingabe des Kommandos kann in einer beliebigen Kombination von Gross- und Kleinbuchstaben erfolgen. In den meisten Faellen kann der erste Parameter ohne Leerzeichen unmittelbar nach dem Kommandonamen folgen.

Die Anzahl der Parameter ist abhaengig vom speziellen Kommando. Benoetigt ein Kommando zwei oder mehr Parameter, muessen sie durch Komma (,) oder Leerzeichen getrennt werden.

6.1. Symbole

Syntax:

<name>

Ein Symbol ist ein <name>, der ein Register, einen Absolutwert, eine Segmentadresse oder einen Segment-Offset repraesentiert. Ein Symbol besteht aus einem oder mehreren Zeichen, beginnend mit einem Buchstaben, einem Unterstreichnungsstrich (_), einem Fragezeichen (?), einem kommerziellen a (@) oder einem Waehrungszeichen ($).

Symbole sind nur fuer den Test verfuegbar, wenn die Symboldatei geladen wurde, in der ihre Namen und Werte definiert sind.

Beachte:

SYMDEB interpretiert die entsprechenden Gross- und Kleinbuchstaben als gleiche Buchstaben. Symbole, die sich durch Gross- oder Kleinschreibung unterscheiden, werden als gleich angesehen. Symbole mit gleichem Namen wie Register, Instruktionen oder Zahlen in hexadezimaler Darstellung werden ignoriert.

Beispiele:

```
_summe
wert_1
start
```

Diese Symbole sind zulaessig, dagegen folgende nicht:

```
ax          ; Register-Name
affe        ; Hexadezimalzahl
add         ; Instruktionsname
```

6.2 Zahlen

Syntax:

```
<ziffern> Y
<ziffern> O
<ziffern> Q
<ziffern> T
<ziffern> H
```

Eine Zahl repraesentiert einen ganzzahligen Wert (integer) und besteht aus einer Kombination von binaeren, oktalen, dezimalen oder hexadezimalen Ziffern und einer wahlfreien Basis. Wenn keine Basis angegeben ist, wird H (hexadezimal) angenommen. Die Basisangabe kann in Gross- oder Kleinbuchstaben erfolgen.

Die folgende Uebersicht zeigt die Ziffern, die bei der entsprechenden Basis benutzt werden koennen:

Basis	Typ	Ziffern
Y	binaer	0 1
O oder Q	oktal	0 1 2 3 4 5 6 7
T	dezimal	0 1 2 3 4 5 6 7 8 9
H	hexadezimal	0 1 2 3 4 5 6 7 8 9 A B C D E F

Hexadezimalzahlen haben den Vorrang vor Symbolen. So wird EFA stets als Hexadezimalzahl interpretiert.

Beispiele:

```
01010111y   127Q   63t   01Hah   efa1
```

6.3 Adressen

Syntax:

```
<segment>:<offset>
```

Eine Adresse ist eine Kombination von zwei 16-Bit-Werten, einer repraesentiert die Segmentadresse, der andere den Segment-Offset.

Eine vollstaendige Adresse besteht aus beiden, Segmentadresse und Offset, getrennt durch einen Doppelpunkt (:).Eine Teil-

adresse besteht nur aus dem Offset.

In beiden Faellen kann oder <offset> eine beliebige Zahl, ein Registername oder ein Symbol sein. Fuer die meisten Kommandos ist die Standard-Segmentadresse der aktuelle Inhalt des DS-Registers.Fuer die Kommandos A, G, L, P, T, U und W hingegen ist dies der Inhalt des CS-Registers.

Adressen koennen als positiver oder negativer Offset eines Symbols angegeben werden.

Beispiele:

```
CS:0100
DS:summe
puffer + 8
```

6.4. Adress-Bereiche

Syntax:

<anfangsadresse> <endadresse>

Ein Adress-Bereich ist ein Adresspaar, das einen zusammenhaengenden Speicherbereich begrenzt, einschliesslich <anfangsadresse> und <endadresse>.

Fordert ein Kommando eine Bereichsangabe und wird die zweite Adresse aber nicht eingegeben, so nimmt SYMDEB einen Bereich von 128 Byte an. Folgt nach dem Bereich ein weiterer Parameter, so muss stets die zweite Adresse eingegeben werden.

Beispiele:

```
summe       summe + 10
DS:200      212
wert_1-20   wert_1
24a         halt
```

6.5. Objekt-Bereiche

Syntax:

<anfangsadresse> L <anzahl>

Ein Objekt-Bereich ist eine Kombination aus Speicheradresse und einer Anzahl von "Objekten", die einen susammenhaengenden Bereich von Byte, Worten, Instruktionen oder anderen Objekten im Speicher bezeichnet. Die <anfangsadresse> spezifiert die Adresse des ersten Objektes und L <anzahl> dessen Anzahl.

Ein Objekt-Bereich kann nur in den Kommandos D, F, S, und U

verwendet werden. Jedes Kommando legt die Groesse und die Art der Objekte fest:

DB hat Byte, DW hat Worte, U hat Instruktionen als Objekte und so weiter.

Beispiel:

```
segm:tabelle1 L 10
```

Wird obiger Bereich im DB-Kommando angegeben, werden die ersten 16 (10h) Byte ab Adresse segm:tabelle1 angezeigt, der gleiche Bereich im U-Kommando bewirkt die Anzeige von 16 Instruktionen ab dieser Adresse.

6.6 Zeichenfolgen

Syntax:

```
'<zeichen>'
"<zeichen>"
```

Eine Zeichenfolge besteht aus ASCII-Zeichen und kann eine beliebige Kombination aus Zeichen sein, die in einfachen (') oder doppelten (") Anfuehrungszeichen eingeschlossen ist. Anfangs- und Ende-Begrenzungszeichen muessen gleich sein. Ein in der Zeichenfolge enthaltenes Begrenzungszeichen muss zweimal angegeben werden.

Beispiele:

```
'Test ist notwendig.'
"Test ist notwendig."
'Dieser "Test" ist notwendig.'
"Dieser ""Test"" ist notwendig."
```

6.7 Ausdruecke

Ein Ausdruck ist eine Kombination aus Parametern und Operatoren und repraesentiert einen 8-, 16-, oder 32-Bit-Wert. Ausdruecke koennen als Werte in Kommandos benutzt werden. Ein Ausdruck kann Symbole, Zahlen oder Adressen mit einem unaeren oder binaeren Operator kombinieren. Unaere Adress-Operatoren nehmen DS als Standard-Segment bei Adressen an, den Registern BP und SP ist jedoch SS als Segment standardmaessig zugeordnet. Ausdruecke werden entsprechend der Prioritaet der Operatoren berechnet, bei gleicher Prioritaet von links nach rechts. Durch die Verwendung von Klammern kann diese Reihenfolge geaendert werden.

Unaere Operatoren

Operator	Bedeutung	Prioritaet
+	unaeres Plus	hoechste
-	unaeres Minus	
NOT	1'er-Komplement	
SEG	Segmentadresse eines Operanden	
OFF	Offset-Adresse eines Operanden	
BY	Byte an Adresse	
WO	Wort an Adresse	
DW	Doppelwort an Adresse	
POI	Zeiger an Adresse (wie DW)	
PORT	1 Byte von einem Port	
WPORT	1 Wort von einem Port	niedrigste

Binaere Operatoren

Operator	Bedeutung	Prioritaet
*	Multiplikation	hoechste
/	ganzzahlige Division	
MOD	Divisionsrest	
:	Segmentadresse	
+	Addition	
-	Subtraktion	
AND	logisches UND	
XOR	Exklusiv-ODER	
OR	logisches ODER	niedrigste

Beispiele:

```
5mod3           ;=2
5/3             ;=1
seg 2000:100    ;=2000
3+(4*5)         ;=23 (17h)
```

7. SYMDEB-Kommandos

Die folgende Tabelle gibt eine Uebersicht ueber die beschriebenen Kommandos:

Kommando	Bedeutung
?	Anzeige
!	Shell Escape

<{	Umlenkung Eingabe	
>}	Umlenkung Ausgabe	
=¯	Umlenkung Eingabe/Ausgabe	
\	Anzeigewechsel	
*	Kommentar	
A	Assemblieren (Assemble)	
BC	Loeschen Unterbrechungspunkt (Breakpoint Clear)	
BD	Unterbrechungspunkt entaktivieren (Breakpoint Disable)	
BE	Unterbrechungspunkt aktivieren (Breakpoint enable)	
BL	Unterbrechungspunkte listen (Breakpoint list)	
BP	Unterbrechungspunkt setzen (Breakpoint Set)	
C	Vergleichen	
D	Speicheranzeige	(Dump)
E	Tastatureingabe	(Enter)
F	Fuellen	(Fill)
G	Echtzeitabarbeitung	(Go)
H	Hexa	(Hex)
I	Port-Eingabe	(Input)
K	Stack-Trace	(Stack-Trace)
L	Laden	(Load)
M	Transport	(Move)
N	Name	(Name)
O	Port-Ausgabe	(Output)
P	PTrace	(PTrace)
Q	Beenden	(Quite)
R	Register	(Register)
S	Suchen	(Search)
T	Trace	(Trace)
U	Reassmblieren	(Unassemble)
W	Schreiben	(Write)
X	Anzeige Symboltabelle	(Examine Symbol Map)
XO	Eroeffnen Symboltabelle	(Open Symbol Map)
Z	Setzen Symbolwert	(Set Symbol Value)

Bei der Eingabe der SYMDEB-Kommandos koennen die speziellen Editor-Tasten benutzt werden. Mit CONTROL-C kann die Ausfuehrung eines SYMDEB-Kommandos abgebro-chen und mit CONTROL-S unterbrochen werden. CONTROL-C und CONTROL-S wirken waehrend der Ausfuehrung des G-Kommandos nur bei einer Ein- oder Ausgabe im zu testenden Programm, sonst ist ein Abbruch nur mit einer Unterbrechungstaste (s. /K) moeglich.

7.1. Assemblieren

Syntax:

A[<adresse>]

Das Assembler-Kommando (A) uebersetzt Instruktionen kompatibel den Instruktionen der 8086-Familie (8086, 8087, 8088, 80186, 80287, 80286-ungeschuetzt) und legt den entstandenen Instruktionscode ab der angegebenen <adresse> ab. Ist keine <adresse> angegeben, wird als Anfangsadresse der aktuelle Inhalt der Register CS und IP genommen.

Vor der Eingabe der Instruktionsmnemonik wird die jeweilige Speicheradresse angezeigt. Die Instruktionen koennen beliebig in Gross- oder/und Kleinschreibung eingegeben werden (die Beispiele verwenden Kleinbuchstaben fuer Instruktionen und Daten und Grossbuchstaben fuer reservierte Woerter).

Um eine neue Instruktion zu uebersetzen, gibt man die gewuenschte Mnemonik ein und bestaetigt die ENTER-Taste. SYMDEB uebersetzt die Instruktion in den Speicher und zeigt die naechste verfuegbare Adresse an. Ein Betaetigen nur der ENTER-Taste beendet das Kommando.

Enthaelt die eingegebene Instruktion einen Syntaxfehler, so zeigt SYMDEB die Meldung ERROR an, wiederholt die aktuelle Assemblier-Adresse und erwartet die Eingabe einer korrekten Instruktion.

Folgende Regeln sind bei der Eingabe von Instruktionen zu beachten:

1. Die Mnemonik bei Rueckkehr aus einem Unterprogramm in einem anderen Codesegment (far return) ist RETF.

2. Die Mnemonik der Instruktionen zur Manipulation von Zeichenfolgen muss explizit eine Groessenangabe enthalten: z.B. MOSV fuer Wort-Zeichenfolge und MOSB fuer Byte-Zeichenfolgen.

3. SYMDEB uebersetzt automatisch Spruenge ueber kurze Distanzen (short), innerhalb eines Segments (near) und in ein anderes Segment (far) abhaengig von der Zieladresse. Dies kann durch Angabe eines Praefix NEAR oder FAR geaendert werden, z.B.:

   ```
   jmp 308
   jmp NEAR 30A
   jmp FAR 310
   ```

   Der NEAR-Praefix kann mit NE abgekuerzt werden, der FAR-Praefix muss vollstaendig angegeben werden.

4. SYMDEB kann nicht unterscheiden, ob sich bestimmte Operanden auf Worte oder Byte beziehen. In diesen Faellen muessen explizit die Praefixe WORD PTR (abgekuerzt: WO) und BYTE PTR (BY) angegeben werden.

   Beispiele:

   ```
   mov WORD PTR[bp],2
   move Byte PTR[si-2],wert1
   ```

5. SYMDEB kann nicht unterscheiden, ob ein Operand eine Speicheradresse oder ein Direktwert ist. Deshalb werden bei SYMDEB Operanden, die sich auf Speicherplaetze beziehen, in eckige Klammern eingeschlossen.

Beispiele:

```
move ax,12     ;Wert 12h wird nach AX transportiert
move ax,[12]   ;Inhalt des Wortes ab 12h wird nach
                  AX transportiert
```

6. Durch die (Pseudo-) Instruktion DB werden Byte-Werte direkt in den Speicher uebersetzt, durch DW Wort-Werte.

   Beispiel:

```
DB 1,2,3,"Beispiel"
DW 1000,2000,'Test'
```

7. SYMDEB akzeptiert alle Darstellungsarten der Register-Indirekt-Adressierung, z.B.:

```
add bx,42[bp+3].[si_2]
pop [bp+di]
push [si]
```

8. Alle synonymen Operationsmnemoniks werden von SYMDEB akzeptiert, z.B.

```
loopz 200
loop 200
ja   300
jnbe 300
```

Beispiel

```
-A
1447:0100 mov ah,2
1447:0102 mov de,7
1447:0104 int 21
1447:0106 mov ah,4c
1447:0108 int 21
1447:010A
-
```

7.2. Unterbrechungspunkte

SYMDEB erlaubt das Setzen und Benutzen von "staendigen" Unterbrechungspunkten. Die fuenf folgenden Kommandos ermoeglichen eine Unterbrechungspunkt-Manipulation:

Kommando	Name
BP	Unterbrechungspunkt setzen
BC	Unterbrechungspunkt loeschen
BD	Unterbrechungspunkt entaktivieren
BE	Unterbrechungspunkt aktivieren
BL	Unterbrechungspunkte listen

7.2.1. Unterbrechungspunkte setzen

Syntax:

BP[<zahl>]<adresse>[<zaehler>][";<kommandos>"]

Mit dem BP-Kommando wird ein "staendiger" Unterbrechungspunkt an adresse definiert. Wird dieser Punkt waehrend der Programmausfuehrung erreicht, so stoppt SYMDEB die Programmausfuehrung und zeigt den aktuellen Wert der Register und die Flageinstellungen im gleichen Format wie das Kommando R an.

Staendige Unterbrechungspunkte, im Gegensatz zu temporaeren Unterbrechungspunkten beim G-Kommando, bleiben so lange wirksam, bis sie entweder geloescht (BC) oder entaktiviert (BE) werden.

Es koennen bis zu 10 Unterbrechungspunkte (0 bis 9) definiert werden. Die <zahl> definiert die Nummer des Unterbrechungspunktes. Leerzeichen zwischen BP und <zahl> sind nicht erlaubt. Wird <zahl> nicht angegeben, wird die erste verfuegbare Nummer zugeordnet. Die Adresse kann eine beliebige Instruktionsadresse sein (d.h. die Adresse des 1.Bytes des Operationscodes der Instruktion).

Der Wert von <zaehler> gibt an, wievielmal der Unterbrechungspunkt beim Erreichen ignoriert wird. Er kann ein beliebiger 16-Bit Wert sein.

Die Kommandos sind eine Folge von SYMDEB-Kommandos, die bei jedem wirksamen Unterbrechungspunkt ausgefuehrt werden. Sie koennen Parameter enthalten und werden durch ein Semikolon (;) getrennt. Die Zeichenanzahl darf 29 nicht uebersteigen.

Beispiele:

```
-BP fehler
-
```

Es wird ein Unterbrechungspunkt an der Adresse fehler gesetzt.

```
-BP2 sub
-
```
Es wird ein Unterbrechungspunkt 2 an Adresse sub gesetzt.

```
-BP 200 4
-
```
Es wird ein Unterbrechungspunkt an Adresse 200 im aktuellen CS-Segment gesetzt. Der Unterbrechungspunkt wird 4-mal vor Wirksamwerden ignoriert.

```
-BP 2871:4309 "db summe summe +3;g"
-
```
Wird die Adresse 2871:4309 erreicht, werden der Inhalt der ersten vier Bytes ab Adresse summe angezeigt und die Programmausfuehrung fortgesetzt.

7.2.2. Unterbrechuogspunkte loeschen

Syntax:

BC <liste>|*

Das BC-Kommando loescht einen oder mehrere vorher gesetzte Unterbrechungspunkte. Ist <liste> eingegeben, so werden die in der Liste enthaltenen Unterbrechungspunkte geloescht (ganzzahlige Werte zwischen 0 und 9).

Bei Angabe von * erfolgt das Loeschen aller Unterbrechungspunkte.

Beispiel:

```
-BC 1 3 4
-
```
Es werden die Unterbrechungspunkte 1, 3 und 4 geloescht.

7.2.3. Unterbrechungspunkte entaktivieren

Syntax:

BD <liste>|*

Das BD-Kommando macht einen oder mehrere Unterbrechungspunkte eines Programms zeitweise unwirksam. Diese Unterbrechungspunkte sind nicht geloescht und koennen mit dem BE-Kommando zu jeder Zeit wieder aktiviert werden.

<liste> enthaelt eine Aufzaehlung von Unterbrechungspunkten (0 bis 9), bei * werden alle Unterbrechungspunkte entaktiviert.

Beispiel:

```
-BD *
-
```

Alle Unterbrechungspunkte werden zeitweise unwirksam.

7.2.4. Unterbrechungspunkte aktivieren

Syntax:

BE <liste>|*

Das BE-Kommanso macht einen oder mehrere durch das BD-Kommando zeitweise nicht wirkenden Unterbrechungspunkte wieder wirksam.

Bei Angabe von <liste> werden die enthaltenen Unterbrechungspunkte (0 bis 9) aktiviert, bei * alle.

Beispiel:

```
-BE 3 4
-
```

Die Unterbrechungspunkte 3 und 4 werden wieder wirksam.

7.2.5. Unterbrechungspunkte listen

Syntax:

BL

Das BL-Kommando listet aktuelle Informationen ueber alle durch das BP-Kommando gesetzten Unterbrechungspunkte.

Das BL-Kommando zeigt die Nummer des Unterbrechungspunktes an, seinen aktuellen Status, die Adresse, den aktuellen Zaehlerstand und den Anfangszaehlerstand (in runden Klammern).

Der Status kann entweder e fuer aktiviert, a fuer entaktiviert oder v fuer virtuell sein. Ein virtueller Unterbrechungspunkt ist ein durch ein Symbol gesetzter Unterbrechungspunkt, dessen EXE-Datei noch nicht geladen wurde.

Sind keine Unterbrechungspunkte gesetzt, erfolgt keine Anzeige.

Beispiel:

```
-BL
4 e 4711:0100  0003  (0005)
6 d 4711:5A73  0002  (000A) "DB 100 102;G"
```

Unterbrechungspunkt 4 auf Adresse 4711:0100 ist aktiviert und wurde zweimal durchlaufen (Anfangszaehlerstand: 5).

Der Unterbrechungspunkt 6 auf 4711:5A73 ist zeitweilig nicht wirksam, er wurde 8-mal durchlaufen und bei seinem Erreichen wurde der Speicherinhalt von 100h bis 102h angezeigt und die Abarbeitung fortgesetzt.

7.3. Kommentar

Syntax:

*** <kommentar>**

Das Kommentar-Kommando besteht aus einem Stern (*), dem der Kommentar-Text folgt. SYMDEB wiederholt den Kommentar-Text auf dem Bildschirm oder einem anderen Ausgabegeraet. Dieses Kommando kann in Verbindung mit den Umlenkungs-Kommandos zum Abspeichern oder Drucken eines SYMDEB-Laufes verwendet werden.

Beispiel:

```
-R DX 100
-* Inhalt von DX auf 100 gesetzt
Inhalt von DX auf 100 gesetzt
-
```

7.4. Vergleichen

Syntax:

C <bereich> <adresse>

Das C-Kommando vergleicht byteweise den Inhalt des durch <bereich> definierten Speicherbereiches mit dem bei <adresse> beginnenden Speicherbereich. Stimmen alle entsprechenden Bytes ueberein, erfolgt keine Anzeige. Bei Nichtuebereinstimmung werden die entsprechenden Bytes paarweise angezeigt.

Beispiel:

```
-C 100,110  200
4270:101  31  00  4270:201
4270:108  0A  10  4270:208
-
```

Hier wird der Bereich von 100h bis 110h mit dem Speicherbereich 200h bis 210h verglichen. Das zweite und neunte Byte besitzen in beiden Bereichen unterschiedliche Werte.

7.5. Anzeige

Syntax:

```
? <ausdruck>
```

Das Anzeige-Kommando (?) berechnet den Wert von <ausdruck> und zeigt ihn in verschiedenen entsprechenden Formaten an: als vollstaendige Adresse, als 16- oder 32-Bit-Hexadezimalzahl, als Dezimalzahl (eingeschlossen in runden Klammern) und einen Zeichenfolgewert (eingeschlossen in doppelten Anfuehrungszeichen), wobei Zeichen mit einem Wert kleiner als 20h oder groesser als 7Eh als Punkt erscheinen.

Der <ausdruck> kann eine beliebige Kombination aus Zahlen, Symbolen, Adressen und Operatoren sein.

Beispiele:

```
-? 7*6
002Ah   0000002A   (42)   "*"
```
Der Wert des Ausdrucks 7 * 6 wird angezeigt.

```
-? DS:summe
42D4:0102h   000H2E42   (273986)   "8"
```
Der Wert der symbolischen Adresse DS:summe wird angezeigt.

```
-? wo ein:wert1
3150h   00003150   (12624)   "1p"
```
Hier wird das Wort an der symbolischen Adresse ein:wert1 angezeigt.

Die Eingabe nur des Fragezeichens (?) bringt als Hilfsmenue eine Uebersicht ueber die Syntax aller Kommandos.

7.6. Speicheranzeige

Syntax:

```
D[<typ>] [<adresse>|<bereich>]
```

Das Kommando bewirkt eine Anzeige des Speicherinhaltes auf dem Bildschirm (oder die Ausgabe auf ein anderes Geraet).

Der <typ> legt das Anzeigeformat fest:

typ	Format
B	Byte
A	ASCII
W	Worte

S	kurze Gleitkommazahlen
L	lange Gleitkommazahlen
T	10-Byte-Gleitkommazahlen

Wird <typ> bei der Kommandoeingabe weggelassen, so erfolgt die Anzeige im Format des zuletzt abgearbeiteten Speicheranzeige-Kommandos. Wurde kein solches Kommando abgearbeitet, erfolgt die Anzeige im Byte-Format (DB). Der Speicherinhalt wird abhaengig von <adresse> oder <bereich> in einer oder mehreren Zeilen angezeigt, am Zeilenanfang stets mit der entsprechenden Speicheradresse.

Werden weder <adresse> oder <bereich> angegeben, so beginnt die Anzeige mit dem Byte, das dem letzten Byte des zuletzt abgearbeiteten Speicheranzeige-Kommandos unmittelbar folgt. Wurde vorher kein Speicheranzeige-Kommando ausgefuehrt, so beginnt die Anzeige mit dem Byte, dessen Adresse dem aktuellen Inhalt von DS:IP entspricht.

Wurde einem vorherigen Speicheranzeige-Kommando kein Segment spezifiert, wird der Inhalt des DS-Registers als Segmentadresse genommen.

Bei der Eingabe des Kommandos ist zu beachten, das D und <typ> unmittelbar ohne Leerzeichen dazwischen aufeinander folgen muessen, dass aber zwischen D[<typ>] und den weiteren Parametern mindestens ein Leerzeichen stehen muss.

7.6.1. Speicheranzeige Byte

Die Werte des spezifierten Speicherbereiches werden als Hexazahlen und ASCII-Zeichen angezeigt. Dabei werden maximal 16 Hexawerte und die entsprechenden ASCII-Zeichen in einer Zeile angezeigt. Die Hexawerte werden ausser dem achten und neunten Wert, zwischen denen ein Minuszeichen (-) als Trennzeichen angezeigt wird, durch je ein Leerzeichen getrennt. Die entsprechenden ASCII-Zeichen folgen lueckenlos aufeinander, dabei werden Bytewerte kleiner als 20h oder groesser als 7Eh als Punkt (.) dargestellt.

Das Kommando (DB) zeigt Werte und Zeichen bis zum Ende von <bereich> an oder 128 Bytes, wenn <bereich> nicht angegeben ist.

Beispiel:

```
-DB DS:100 108
1447:0100 3D 02 00 74 2A 3D 03 00-74   =..t*=..t
```

7.6.2. Speicheranzeige ASCII

Die Anzeige des Inhaltes des angegebenen Speicherbereiches erfolgt in ASCII-Zeichen. In einer Zeile koennen bis zu 48 Zeichen angezeigt werden. Bytes, deren Wert kleiner als 20h oder groesser als 7Eh ist, werden als Punkt (.) dargestellt.

Ist <adresse> angegeben, erfolgt die Anzeige ab <adresse> bis zum Erreichen eines Bytes mit dem Wert 00h, jedoch nicht mehr als 128 Zeichen. Ist <bereich> angegeben, erfolgt die vollstaendige Anzeige des Inhaltes des Speicherbereiches.

Sind weder <adresse> noch <bereich> angegeben, erfolgt die Anzeige ebenfalls entweder bis zum ersten Bytewert 00h oder von 128 Zeichen, wenn unter ihnen kein solcher Wert auftritt.

Beispiel:

```
-DA DS:100 108
1447:0100 = ..t*=..t

-D L 2
1447:0109 PK
-
```

7.6.3. Speicheranzeige Worte

Das Kommando (DW) zeigt die hexadezimalen Werte der Worte (2-Byte- Werte) ab <adresse> oder in dem angegebenen <bereich> an, dabei werden maximal 8 Worte in einer Zeile angezeigt, die durch je ein Leerzeichen getrennt werden.

Ist <bereich> nicht angegeben, werden 64 Worte angezeigt.

Beispiel:

```
-DW DS:100 108
1447:0100 023D 7400 3D2A 0003 5074
-
```

7.6.4. Speicheranzeige Doppelworte

Das Kommando (DD) zeigt den Speicherinhalt ab <adresse> oder im <bereich> als Doppelworte (4-Byte-Werte) hexadezimal an. In einer Zeile werden maximal 4 Doppelworte angezeigt, jeweils durch ein Leerzeichen getrennt. Zwischen den beiden Worten eines Doppelwortes erfolgt die Anzeige eines Doppelpunktes (:).

Ist <bereich> angegeben, werden 32 Doppelworte angezeigt.

Beispiele:

```
-DD DS:100 L 3
1447:0100 7400:023D 0003:3D2A 0FEB:5074
```

Hier besteht <bereich> aus drei Doppelworten.

7.6.5. Speicheranzeige Gleitkommazahlen

Die Anzeige des Speicherinhaltes hexadezimal und dezimal als Gleitkommazahl erfolgt mit den Kommandos DS (4 Byte), DL (8 Byte) und DT (10 Byte).

Der Dezimalwert hat die Form

+|- 0.<dezimalziffern>E +|- <mantisse>

Das der Null folgende Zeichen ist der Dezimalpunkt (.). Daran schliessen sich maximal 16 Dezimalziffern an (beim Kommando DS sind davon nur 7 signifikant). Die Mantisse beginnt mit dem Buchstaben E, gefolgt vom Vorzeichen und den Ziffern (<mantisse>).

Je Zeile wird ein Gleitkomma angezeigt, wird kein <bereich> im Kommando angegeben, erfolgt die Anzeige von genau einer Zahl.

Beispiele:

```
-DS DS:100 108
1447:0100 3D 02 00 74 +0.405675900652819E+32
1447:0104 2A 3D 03 00 +0.2974480198283716E-39
1447:0108 74 50 EB 0F +0.2320377850737863E-28

-DL DS:0100 108
1447:0100 3D 02 00 74 2A 3D 03 00 +0.45o4285200946604E-308
1447:0108 74 50 EB 0F E8 35 DD A3 -0.6279456170415646E-135

-DT DS:100 108
1447:0100 3D 02 00 74 2A 3D 03 00 74 50 +0.0001715233419558
E+1268

-D
1447:010A EB 0F E8 35 DD A3 12 20 0B C0 -0.1026330011769433
E+4
-
```

7.7. Tastatureingabe

Syntax:

E[<typ>] <adresse> [<liste>|<wert>]

Das Kommando ermoeglicht die Eingabe von Werten in verschiedenen Formaten von der Tastatur (oder einem anderen Eingabegeraet) in den Speicher.

Das Format der Eingabewerte wird von <typ> festgelegt und entspricht dem der Speicheranzeige (D). Erfolgt keine Angabe von <typ>, wird das Format im zuletzt ausgefuehrten Eingabebefehl angenommen bzw. EB beim ersten Eingabebefehl.

Durch eine syntaktisch fehlerhafte Eingabe wird der Speicherinhalt nicht veraendert.

Eine Eingabe von <liste> ist nur bei EB und EA moeglich.

Erfolgt keine Angabe von Werten (<liste> bzw. <wert>), so werden Adresse <adresse> und aktueller Wert gefolgt von einem Punkt angezeigt. Danach kann die Eingabe eines neuen Wertes im entsprechenden Format erfolgen oder das Kommando durch Druecken von ENTER ohne Werteingabe beendet werden.

7.7.1. Eingabe Byte

Mit EB werden ein oder mehrere Bytewerte ab <adresse> in den Speicher geschrieben. Wurden weder <liste> noch <wert> eingegeben, koennen byteweise ein neuer Wert eingegeben, zum naechsten Byte ohne Eingabe uebergegeangen oder zu einem vorhergehenden Byte zurueckgekehrt werden.

- Die Eingabe des neuen Wertes erfolgt nach der Anzeige des aktuellen Wertes.

- Durch Betaetigen der LEER-Taste erfolgt der Uebergang zum naechsten Byte. Nach jeweils 8 Byte erfolgt die Fortsetzung der Anzeige auf der naechsten Bildschirmzeile.

- Zu einem vorhergehenden Wert kann durch Eingabe des Minuszeichens (-) zurueckgekehrt werden.

Beispiele:

```
-EB DS:100 12 34
-DB DS:100 L 2
1447:0100 12 34                                  .4
-
```

Es werden zwei Byte in den Speicher eingegeben und der betreffende Speicherbereich mit DB angezeigt.

```
-EB DS:100
1447:0100  31.45  32.  00.-
1447:0101  32.67  00.

-DB DS:100 L 3
1447:0100  45  67  00                          Eg.
-
```

Im zweiten Beispiel wird der Wert des ersten Bytes geaendert, das zweite Byte durch Betaetigen der LEER-Taste unveraendert gelassen. Durch die Eingabe des Minus-Zeichens wird zum vorhergehenden Byte zurueckgekehrt, der Wert geaendert und das Kommando beendet.

7.7.2. Eingabe ASCII

Das EA-Kommando ist identisch mit dem EB-Kommando.

Beispiel:

```
-EA DS:100 "AB"
-DB DS:100 L 2
1447:0100 41 42                          AB

-EA DS:100
1447:0100 41.  42.43  00.

-DB DS:100 L 3
1447:0100 41 43 00                       AC.
```

7.7.3. Eingabe Wort

Das EW-Kommando schreibt einen Wort-Wert in den Speicher. Der wahlweise angebbare <wert> besteht aus einem Wort.

Wird <wert> nicht angegeben, zeigt das Kommando den aktuellen Wert des Wortes ab <adresse> an. Es koennen ein neuer Wert eingegeben und zum naechsten Wort uebergegangen oder mit ENTER die Kommandoausfuehrung beendet werden.

Beispiel:

```
-EW DS:100 1234
1447:0102 0000.5678
1447:0104 0000.

-DB DS:100 L 4
1447:0100 34 12 78 56                    4.xV
-
```

7.7.4. Eingabe Doppelwort

Das Kommando ED wirkt wie EW, die beiden Worte eines Doppelwortes sind durch einen Doppelpunkt zu trennen.

Beispiel:

```
-ED DS:100
1447:0100 5678:1234.9876:5432
1447:0104 0000:0000.

-DB DS:100 L 8
1447:0100 32 54 76 98                    2Tv.
-
```

7.7.5. Eingabe Gleitkommazahlen

Die als Dezimalzahlen eingegebenen Werte werden in 4 Byte (ES), 8 Byte (EL) oder 10 Byte (ET) in den Speicher geschrieben. Die Anzeige des aktuellen Inhalts erfolgt hexadezimal und dezimal in Gleitkommadarstellung.

Beispiele:

```
-ES DS:100 3.14
-DS DS:100
1447:0100 C3 F5 48 40 +0.31400001049041750E+1

-ET DS:100 1.23456789e-1
-DT DS:100
1447:0100 89 F9 1A CB B9 E9 D6 FC FB 3F +0.123456789E+0
-
```

7.8. Anzeige Symboltabelle

Syntax:

```
X[*]
X?[<tabellenname>!] [<segmentname>:] [<symbolname>]
```

Das Kommando X bzw. X? zeigt die Namen und Adressen der Symbole der Symboltabellen an. SYMDEB bildet eine Symboltabelle fuer jede Symboldatei, deren Name in der SYMDEB-Kommandozeile angegeben und geladen ist.

X zeigt Namen und Segmentadressen der Segmente der aktuellen (eroeffneten) Symboltabelle an, bei X* von allen gebildeten Symboltabellen.

X? zeigt Namen und Adressen eines oder mehrerer Symbole der aktuellen Symboltabelle an bzw. bei <tabellenname>! die der angegebenen Tabelle. Der <tabellenname> muss der Name einer geladenen Symboldatei (ohne Erweiterung) sein. Dem Namen folgt ein Ausrufezeichen (!).

Ist <segmentname>: angegeben, werden Name und Adresse dieses Segmentes angezeigt, das entweder in der aktuellen oder explizit angegebenen Symboltabelle liegen muss. Dem Segmentnamen muss ein Doppelpunkt (:) folgen.
Wird ein <symbolname> angegeben, werden Segmentadresse und Offset dieses Symbols, das im spezifierten Segment existieren muss, angezeigt.
Sollen Informationen ueber mehr als ein Segment oder Symbol angezeigt werden, kann ein Teil eines Segment- oder Symbolnamens, gefolgt von einem Stern (*) bzw. nur ein Stern eingegeben werden.

Fuer die nachfolgenden Beispiele wird vorausgesetzt, dass zwei Symboldateien mit der SYMDEB-Kommandozeile

```
SYMDEB test1.sym test.sym test.exe
```

geladen wurden.

Beispiele:

```
-X
[3942 TEST]
     3952 DATEN
     [39D6 CODE]
-
```

Hier werden der Name der aktuellen Symboltabelle und die Namen und Segmentadressen der Segmente in dieser Tabelle angezeigt. Die Klammern zeigen an, dass eine Tabelle oder ein Segment eroeffnet ist.

Ein eroeffnetes Segment wird bei einem SYMDEB-Kommando, das ein Symbol enthaelt, zuerst nach diesem Symbol durchsucht und ermoeglicht so einen schnellen Zugriff.

```
-X?test1!
2448 TEST1
-
```

Die Segmentadresse der Tabelle TEST1 wird angezeigt.

```
-X?test!daten:su*
DATEN: (3952)
3961 SUMME    3984 SUCHB
-
```

Die Adressen aller mit SU beginnenden Symbole im Segment DATEN der Tabelle TEST werden angezeigt.

7.9. Fuellen

Syntax:

F<bereich> <liste>

Das Kommando schreibt die in <liste> angegebenen Werte solange in den durch <bereich> definierten Speicherbereich, bis dieser gefuellt ist.

Beispiel:

```
-F DS:100 L 5 31 32
-DB DS:100 L 6
1447:0100 31 32 31 32 31 00        12121.
-
```

7.10. Echtzeitabarbeitung

Syntax:

G[=<startadresse>][<stoppunkte>]

Das G-Kommando startet die Abarbeitung des zu testenden Programmes ab der angegebenen <startadresse>, ist diese nicht explizit angegeben, bei der Adresse entsprechend dem aktuellen Inhalt der Register CS und IP. Die Abarbeitung wird beendet, wenn entweder das Programmende oder einer der angegebenen <stoppunkte> erreicht ist.

Beim Erreichen eines der durch das Kommando BP gesetzten und aktivierten Unterbrechungspunkte wird ebenfalls die Abarbeitung unterbrochen und damit die Kommandoausfuehrung beendet.

Die Stoppunkte beim G-Kommando sind temporaere Unterbrechungspunkte, d.h. sie sind nur wirksam waehrend der Ausfuehrung des G-Kommandos, in dem sie definiert wurden. Man kann bis zu 10 Stoppunkte in beliebiger Reihenfolge im Kommando angeben.

Ist ein Stoppunkt erreicht, werden die aktuellen Inhalte der Register und Flags entsprechend dem R-Kommando angezeigt.

Beachte:

SYMDEB benutzt bei Ausfuehrung des G-Kommandos eine IRET-Instruktion. Deshalb werden der Inhalt des Flag-Registers und der der Register CS und IP in den Anwender-Stack gerettet. Sind dort nicht mindestens 6 Bytes verfuegbar, erfolgt ein Systemabsturz. SYMDEB schreibt eine INT-Instruktion (Interrupt mit dem Code 0CCh) an jede Stoppunkt-Adresse und ersetzt nach dem Erreichen eines Unterbrechungspunktes diese durch die vorherigen Werte. Dies erfolgt nicht, wenn das Programmende vor einem ge-

setzten Unterbrechungspunkt erreicht wird. Daher sollte vor Testfortsetzungen mit den Kommandos N und L das zu testende Programm wieder geladen werden.

SYMDEB zeigt "Program terminated normally" an, wenn waehrend der Abarbeitung das Programmende erreicht wurde. Die Abarbeitung wird beendet und die aktuellen Flag- und Registerwerte angezeigt.

Beispiel:

```
-G = anfang druck
```

SYMDEB startet die Programmausfuehrung mit der an der symbolischen Adresse anfang beginnenden Instruktion und beendet das Kommando nach dem Erreichen des Programmendes oder der bei Adresse druck beginnenden Instruktion oder eines durch BP gesetzten Unterbrechungspunktes.

7.11. Hexa

Syntax:

```
H <wert1> <wert2>
```

Das H-Kommando zeigt die Summe <wert1> + <wert2> und die Differenz <wert1> - <wert2> zweier Hexadezimalzahlen an.

Beispiel:

```
-H ab 3 4
0AB7 0AAF
-
```

7.12. Port-Eingabe

Syntax:

```
I<port>
```

Das Kommando liest ein Byte vom angegebenen port (beliebige 16-Bit-Portadresse) und zeigt seinen Wert an.

Beispiel:

```
-I 2f6
E6
-
```

7.13. L-Kommando

Syntax:

L[<adresse>[<laufwerk> <satz> <anzahl>]]

Das Kommando kopiert den Inhalt einer Datei oder einer bestimmten Anzahl von logischen Saetzen ab <adresse> oder, wenn keine Adresse angegeben ist, ab CS:100 in den Speicher.

Das Registerpaar BX:CX enthaelt die Anzahl der kopierten Bytes.

Vor dem Laden einer Datei muss ihr Name festgelegt werden. Das kann mit dem N-Kommando oder als Argument beim Laden von SYMDEB erfolgen. Wurde auf diese Weise kein Name festgelegt, so nimmt SYMDEB den aktuellen Inhalt des Standard-Dateikennblockes ab DS:5C als Name. Das Lesen von logischen Saetzen von einer Diskette erfordert die Angabe von Laufwerk <laufwerk>, Satz-Nummer <satz> und Anzahl der logischen Saetze <anzahl>.

Das <laufwerk> ist eine Zahl im Bereich 0 bis 3, die das Laufwerk A(0), B(1), C(2) oder D(3) repraesentiert.

<satz> und <anzahl> sind jeweils 1- bis 4-stellige Hexadezimalzahlen.

Beachte:

Hat die geladenen Datei die Erweiterung .EXE, kopiert das L-Kommando die Datei ab der im Dateikennsatz spezifierten Ladeadresse. Eine Angabe von <adresse> wird ignoriert.

Beispiele:

```
-N test.exe
-L
-
```

Die Laenge der Datei test.exe (minus Laenge des Dateikennsatzes) wird von SYMDEB in das Registerpaar BX:CX eingetragen.

```
-L menue 1 12 H
-
```

Vier logische Saetze von der Diskette in Laufwerk B, beginnend ab logischer Satznummer 12h werden in den Speicher ab der symbolischen Adresse menue geladen.

7.14. Transport

Syntax:

M <bereich> <adresse>

Das Kommando transportiert den durch <bereich> spezifierten zusammenhaengenden Speicherbereich in den bei <adresse> beginnenden Speicherbereich.

Der Zielbereich ist stets eine vollstaendige Kopie des Quellbereichs, auch bei Ueberlagerung beider Bereiche.

Beispiele:

```
-M ds:100 107 ds:204
-M summe 1 8 summe + 20
-
```

7.15. Name

Syntax:

N [<dateiname>][<argumente>]

Mit dem N-Kommando werden ein Dateiname fuer ein nachfolgendes L- oder W-Kommando und Argumente fuer die Ausfuehrung eines geladenen Programms gesetzt.

Ist <dateiname> angegeben, benutzen alle nachfolgenden L- und W-Kommandos diesen Namen beim Disketten-Zugriff.

Werden <argumente> spezifiert, kopiert das Kommando alle Argumente einschliesslich Leerzeichen in den bei DS:81 beginnenden Speicherbereich und traegt in DS:80 die Anzahl der kopierten Zeichen ein. Die Argumente stehen dann dem zu testenden Programm zur Verfuegung.

Beachte:

Sind die ersten beiden <argumente> ebenfalls Dateinamen, erzeugt das Kommando Dateisteuerbloecke (FCB's) ab DS:5C und DS:6C und kopiert die Namen in diese Bloecke. Die Dateisteuerbloecke koennen dann vom zu testenden Programm benutzt werden.

Das N-Kommando behandelt auch <dateiname> als Argument, kopiert ihn ab DS:81 und erzeugt einen Dateisteuerblock ab DS:5C. Deswegen loescht das Definieren eines neuen Dateinamens fuer die L- und W-Kommandos vorher angegebene Programmargumente.
Jedes N-Kommando aendert einen oder mehrere der folgenden Speicherbereiche:

Adresse	Inhalt
DS:5C	Dateisteuerblock fuer Datei 1
DS:6C	Dateisteuerblock fuer Datei 2
DS:80	Zeichenanzahl
DS:81	eingegebene Zeichen

Beispiel:

```
-N test.exe
-D 80 8a
4662:0080 09 20 74 65 73 74 2E 65 78 65      .test.exe
-
```

7.16. Eroeffnen Symboltabelle

Syntax:

```
XO [<tabellenname>!][<segmentname>]
```

Das XO-Kommando setzt (eroeffnet) die aktuelle Symboltabelle und/oder das aktuelle Segment.

Ist <tabellenname> angegeben, eroeffnet SYMDEB die angegebene Tabelle. <tabellenname> muss der Name (ohne Erweiterung) einer in der SYMDEB-Kommandozeile angegebenen Symboldatei sein.

<segmentname> muss der Name eines Segments in der aktuellen Symboltabelle sein. Alle Segmente der Symboltabelle sind natuerlich verfuegbar, das aktuelle Segment wird aber zuerst nach einem Symbol durchsucht.

Beispiel:

```
SYMDEB test1.sym test.sym test.exe
-X*
2448 TEST1
     2458 DATEN
     2662 CODE
[3942 TEST]
     3952 DATEN
     [39D6 CODE]
-XO test1!daten
-X*
[2448 TEST1]
     [2458 DATEN]
     2662 CODE
3942 TEST
     3952 DATEN
     39D6 CODE
-
```

7.17. Port-Ausgabe

Syntax:

```
O <port> <byte>
```

Das angegebene <byte> wird zum festgelegten <port> (16-Bit-Port-Adresse) gegeben.

Beispiel:

```
-O 2f8 4f
-
```

7.18. PTrace

Syntax:

```
P[=<startadresse>][<anzahl>]
```

Das P-Kommando fuehrt die an <startadresse> beginnenden Instruktionen aus und zeigt dann die aktuellen Werte aller Speicher und Flags an. Die Anzeige erfolgt im gleichen Format wie beim R-Kommando.

Ist keine <startadresse> angegeben, wird der aktuelle Inhalt der Register CS und IP genommen.

Ist <anzahl> spezifiert, fuehrt SYMDEB vor einem Stop die entsprechende Anzahl von Instruktionen aus, die Anzeige der aktuellen Register- und Flagwerte erfolgt nach jeder Instruktionsausfuehrung.

Beachte:

Das PTrace-Kommando (P) wirkt wie das Trace-Kommando (T), ausser dass es beim Erreichen von Unterprogramm-Aufrufen (call) oder Software-Interrupts diese bis zur Rueckkehr in das aufrufende Programm im Echtzeitbetrieb abarbeitet, waehrend das T-Kommando diese Programmzeile ebenfalls schrittweise ausfuehrt.

Aber weder das P- noch das T-Kommando erlauben die schrittweise Abarbeitung des Interrupts 21h (Betriebssystem-Funktionsrufe).

Beispiele:

```
-P = druck
AX=0400  BX=0003  CX=0400 DX=001A SP=01FE  BP=0000  SI=0018
DI=0000
```

```
DS=1447  ES=1447 SS=1862 CS=1447 IP=0108 NV UP EI PL NZ  NA
PE NC
1447:0108 B745          MOV BH,45                ;'E'
-
```

Es wird die bei der Adresse druck beginnende Instruktion ausgefuehrt und die nach der Ausfuehrung aktuellen Registerwerte und Flageinstellungen sowie Adresse, Code und Mnemonik der naechsten auszufuehrenden Instruktion angezeigt.

7.19. Beenden

Syntax:

```
Q
```

Das Q-Kommando beendet SYMDEB und kehrt zum Betriebssystem zurueck.

7.20. Umlenkung

Syntax:

```
< <geraetename>
> <geraetename>
= <geraetename>
{ <geraetename>
} <geraetename>
- <geraetename>
```

Die Umlenkungs-Kommandos bewirken, dass sowohl die SYMDEB-Kommandoein- und -ausgabe als auch die EIN- und Ausgaben im zu testenden Programm von dem bzw. auf das als <geraetename> spezifierte Geraet erfolgen:

	SYMDEB-Kommandos	zu testendes Programm
Eingabe	<	{
Ausgabe	>	}
Ein- und Ausgabe	=	-

<geraetename> kann ein beliebiger DCP-Geraetename oder Dateiname sein. Ein typischer Anwendungsfall der Umlenkungs-Kommandos ist der Test von Programmen, die viele Bildschirmausgaben besitzen. So kann man z.B. die Ausgaben des zu testenden Programmes auf einen Farbgrafik-Monitor umlenken, waehrend die Kommando-Anzeige auf einem Schwarz-Weiss-Monitor erfolgt.

Beachte:

Werden die Eingaben auf COM1 oder COM2 umgelenkt, sind die Tastenkombinationen CONTROL-S (Unterbrechung Kommando) und CONTROL-C (Abbruch Kommando) unwirksam.

Beispiel:

-< symkdo.txt
Die Kommandoeingabe erfolgt nicht mehr ueber die Tastatur, sondern von der Datei symkdo.txt. Enthaelt diese Datei eine Folge von SYMDEB-Kommandos (durch 0Dh: (carriage return)) getrennt, fuehrt SYMDEB diese Kommandos bis zum Dateiende aus. Die letzten Kommandos in dieser Datei sollten Q oder <CON sein, da sonst keine Moeglichkeit besteht, SYMDEB das Ende des Testlaufes mitzuteilen.

7.21. Register

Syntax:

R[<registername>[[=]<wert>]]

Das Register-Kommando R zeigt die Inhalte der CPU (Central Processing Unit)-Register an und ermoeglicht ihre Aenderung.

Wird ein <registername> angegeben, zeigt das Kommando die Werte aller Register und Flags und die Instruktion an, deren Adresse dem Inhalt von CS und IP entspricht.

Das Trace (T)- und das PTrace (P)-Kommando zeigen die Register im gleichen Format wie das R-Kommando an.

Wird <registername> spezifiert, zeigt das Kommando den aktuellen Registerwert an und erwartet nach der Anzeige eines Doppelpunktes (:) die Eingabe eines neuen Wertes. Soll der Wert unveraendert bleiben, muss die ENTER-Taste gedrueckt werden. Sind <registername> und <wert> angegeben wird der Registerinhalt auf den spezifierten Wert geaendert.

Als <registername> kann einer der folgenden Namen angegeben werden: AX, BX, CX, DX, CS, DS, SS, ES, SP, BP, SI, DI, IP, PC oder F.

IP und PC bezeichnen das gleiche Register: den Befehlszaehler (Instruction Pointer).
F ist der Name des Flag-Registers. Die anderen Registernamen entsprechen denen der vom Assembler verwendeten.

Bei Angabe eines ungueltigen Registernamens erfolgt die Ausschrift "Bad Register!" durch SYMDEB.

Um einen Flag-Wert zu aendern, gibt man den Registernamen F ein und erhaelt die Anzeige der aktuellen Werte als Name:

Flag	gesetzt	geloescht
Ueberlauf (Overflow)	OV	NV
Richtung	DN (abwaerts)	UP (aufwaerts)
Interrupt	EI (moeglich)	DI (nicht moeglich)
Vorzeichen (Sign)	NG (negativ)	PL (positiv)
Null (Zero)	ZR	NZ
Hilfsuebertrag (Auxiliary Carry)	AC	NA
Paritaet (Parity)	PE (gerade)	PO (ungerade)
Uebertrag (Carry)	CY	NC

Am Ende der Liste zeigt das Kommando ein Minuszeichen (-) an und die Eingabe der neuen Werte kann erfolgen. Die Werte koennen in beliebiger Reihenfolge eingegeben werden, Leerzeichen zwischen ihnen sind nicht erforderlich. Die Eingabe wird durch die ENTER-Taste beendet. Werden fuer ein Flag zwei Werte eingegeben (z.B. OV NV), erfolgt die Fehleranzeige "Double Flag!" und bei einer ungueltigen Wertebezeichnung "Bad Flag!". In beiden Faellen werden nur die Flagwerte geaendert, die bis zum Fehler eingegeben wurden.

Beispiele:

```
-R AX
AX 0000
:1234
-
```

Der neue Wert von AX ist 1234h.

```
-R BX 3
-
```

Als neuen Wert erhaelt BX 0003h.

```
-R F
NV UP EI NZ NA PO NC - pecy
-R
AX=1234  BX=0003  CX=0000 DX=0000 SP=EAA8  BP=0000  SI=0000
D=0000  DS=1447 ES=1447 SS=1447 CS=1447 IP=0100 NV UP EI PL
NZ NA PE CY
1447:0100 0000    ADD   [BX+SI],AL    DS:0002=A0
-
```

Bei der Anzeige des naechsten Befehls wird der dem Operanden entsprechende Speicherwert angezeigt, d.h. der Speicherplatz BX+SI als Offset des DS-Segmentes enthaelt den Wert A0h.

7.22. Anzeigewechsel

Syntax:

\

Das Anzeigewechsel-Kommando (\) erlaubt den Wechsel der Anzeige (screen swap) von SYMDEB auf die des zu testenden Programmes. Durch Betaetigung einer beliebigen Taste wird zur SYMDEB-Anzeige zurueckgekehrt.

Diese Moeglichkeit ist besonders beim Test von anzeigeintensiven Programmen, zum Beispiel Grafik-Programmen, von Bedeutung.

Das Kommando ist nur wirksam, wenn die /S-Option beim Aufruf von SYMDEB angegeben wurde.

7.23. Suchen

Syntax:

S<bereich> <liste>

Das Such-Kommando (S) durchsucht den als <bereich> angegebenen Speicherbereich nach den in <liste> angegebenen Werten. Werden die entsprechenden Werte im Speicher gefunden, erfolgt die Anzeige ihrer Speicheradresse, ansonsten wird nichts angezeigt. <liste> kann eine beliebige Zahl von Bytes enthalten, die durch ein Leerzeichen oder Komma getrennt sind. Bei mehr als einem Byte wird der Speicher nach dem Auftreten der angegebenen Bytefolge lueckenlos und in gleicher Anordnung durchsucht.

Beispiele:

```
-S text 1 100 "Bild"
1447:0632
1447:0708
-
```

Ab Adresse text werden 256 Bytes (100h) nach der Zeichenfolge "Bild" durchsucht. Die Speicheradressen, an denen diese Zeichenfolge im durchsuchten Speicherbereich auftritt, werden angezeigt.

```
-S cs:100 17f 1a
1447:0142
-
```

Ab CS:0100 werden 128 Byte nach dem Codezeichen 1Ah durchsucht und an Adresse CS:0142 gefunden.

7.24 Shell_Escape

Syntax:

```
!<kommando>
```

Das Shell-Escape-Kommando (!) ermoeglicht das Ausfuehren von COMMAND.COM und DCP-Kommandos innerhalb von SYMDEB. Das Shell-Kommando ! allein fuehrt COMMAND.COM ohne Argumente aus, die aktuellen Werte des zu testenden Programmes werden gerettet. Nach Ausfuehrung der gewuenschten DCP-Kommandos kann mit dem DCP-Kommando EXIT an die Stelle in SYMDEB nach Eingabe des Shell-Escape-Kommandos zurueckgekehrt werden.

Zusaetzlich kann ein DCP-Kommando oder der Name eines Programmes direkt nach der Kommando-Bezeichnung ! angegeben werden. Das Kommando wird automatisch ausgefuehrt und nach Beenden zu SYMDEB zurueckgekehrt.

Beachte:

Um das Shell-Escape-Kommando benutzen zu koennen, muss das zu testende Programm den nicht benoetigten Speicherplatz freigeben. Ein Programm kann dies durch den DCP-Funktionsaufruf (INT 21h) 4Ah (Modify Allocate Memory). Damit steht DCP Speicherbereich zum Laden der neuen Datei COMMAND.COM zur Verfuegung. Das Gleiche kann durch die /CPARMAXALLOC-Option beim LINK-Lauf erreicht werden.

Wurde kein Speicherbereich freigegeben, so zeigt SYMDEB durch die Ausschrift "Not enough memory" an, dass das Shell-Escape-Kommando nichr ausgefuehrt werden kann.

Der gesamte Text nach der Shell-Escape-Kommandobezeichnung (!) wird als DCP-Kommandozeile interpretiert.

SYMDEB benutzt zum Abspeichern einer Kopie von COOMAND.COM die Umgebungs (Environment)-Variable COMSPEC.

Beispiel:

```
-!dir a:test3.*
Dskt/Platte in Laufwerk A hat keinen Namen
Verzeichnis von A:\WORK
TEST3.CRF       67          18.05.87        7.10
TEST3.ASM       384         18.05.87        8.30
          2 Datei(en)       131072 Byte frei
-
```

Das interne DCP-Kommando DIR wird ausgefuehrt, seine Ausgabe erfolgt auf dem Bildschirm und die Steuerung wird an SYMDEB zurueckgegeben.

7.25. Stack Trace

Syntax:

K[<anzahl>]

Das Stack-Trace-Kommando (K) ermoeglicht die Anzeige der aktuellen Stack-Struktur. Die erste Anzeigezeile enthaelt den Namen der aktuellen Prozedur und den Namen der die Prozedur aufrufenden Prozedur. Die folgenden Zeilen, falls vorhanden, tracen den Aufruf. So enthaelt z.B. die naechste Zeile Namen und Argumente der die aktuelle Prozedur aufrufenden Prozedur usf.

SYMDEB zeigt die Argumente einer Prozedur nur an, wenn ihre Anzahl bekannt ist. Das kann explizit durch die Angabe von <anzahl> erfolgen, die die Argumentanzahl als Anzahl von Werten festlegt.

Beachte:

Das Stack-Trace-Kommando ist nur bei solchen Assemblerprogrammen anwendbar, deren Prozedur-Aufruf bestimmten Regeln genuegt, das sind Regeln, die bei hoeheren Programmsprachen zur Anwendung kommen (Parameteruebergabe im Stack). Ein Beispiel zeigt einen solchen Prozedur-Aufruf:

```
          .
          .
          .
        push ax             ; 2. Argument
        push bx             ; 1. Argument
        call bsp            ; Aufruf Prozedur
        add sp,4            ; Ruecksetzen Stackpointer
          .
          .
          .
    bsp  PROC near
     a1: push bp
        mov  bp,sp
        mov  ax,[bp+4]      ; Laden 1. Argument
        mov  bx,[bp+6]      ; Laden 2. Argument
          .
          .
          .
        pop bp
        ret
    bsp  ENDP
```

Beispiel:

```
-K 2
Test:A1(0002,0001)
-
```

A1 wird mit zwei Argumenten, die die aktuellen Werte 2 und 1 besitzen, aufgerufen.

7.26. Setzen Symbolwert

Syntax:

Z<symbol> <wert>

Das Z-Kommando setzt die aktuelle Adresse des angegebenen Symbols auf den spezifierten Wert.

Beispiel:

```
-Z summe 9d
-
```

Die Adresse des Symbols summe erhaelt den Wert 9Dh.

7.27. Trace

Syntax:

T[=<startadresse>][<anzahl>]

Das Trace-Kommando wirkt wie das PTrace-Kommando (P), nur werden beim T-Kommando durch CALL aufgerufene Unterprogramme und Interrupts ebenfalls schrittweise abgearbeitet, ausser DCP-Funktionsrufe (INT 21h). Das Trace-Kommando benutzt den Hardware-Trace-Modus des Prozessors, daher koennen auch im ROM (Read-Only-Memory) gespeicherte Instruktionen mit dem Kommando abgearbeitet werden.

7.28. Reassemblieren

Syntax:

U[<bereich>]

Das Reassemblier-Kommando U zeigt die Instruktionen des zu testenden Programmes im angegebenen Bereich an.

Ist kein <bereich> spezifiert, werden beginnend an der aktuellen Reassemblier-Adresse acht Instruktionen angezeigt. Die aktuelle Reassemblier-Adresse ist die Adresse des Bytes, das dem zuletzt angezeigten Byte des vorhergehenden U-Kommandos unmittelbar folgt.

SYMDEB zeigt sowohl den Hexadezimalwert (als Teil der Instruktion) als auch die ASCII-Darstellung (als Kommentar nach einem Semikolon) von 8-Bit-Direktwert (immediate)-Operanden an.

Beispiel:

```
-U 100 108
1445:0100 B402      MOV AH,02
1445:0102 B207      MOV DL,07
1445:0104 CD21      INT 21
1445:0106 B44C      MOV AH,4C
1445:0108 CD21      INT 21
-
```

7.29. Schreiben

Syntax:

W[<adresse>[<laufwerk> <satz> <anzahl>]]

Das W-Kommando schreibt den Inhalt eines Speicherbereiches als Datei oder logischen Satz auf Diskette.

Beim Schreiben als Datei muessen der Name der Datei mit einem N-Kommando festgelegt worden sein und das Registerpaar BX:CX die Anzahl der auszugebenden Bytes enthalten.

Die Anfangsadresse des auszugebenden Speicherbereiches wird explizit als <adresse> angegeben, ansonsten wird CS:100 als Adresse angenommen, wobei CS der aktuelle Inhalt des CS-Registers ist.

Erfolgt die Ausgabe in logischen Saetzen, muessen Adresse, Laufwerk (A=0, B=1, C=2, D=3), Nummer des ersten logischen Satzes (1- bis 4-stellige Hexadezimalzahl) und Anzahl der logischen Saetze (1- bis 4-stellige Hexadezimalzahl) angegeben werden.

Beispiel:

```
-N b:test.com
-R BX 00
-R CX 12
-W 200
-
```

Ab Adresse 200h werden 18 Byte (12h) als Datei TEST.COM auf die sich im Laufwerk B befindliche Diskette geschrieben.

```
-W text1 0 27 2
-
```

Der Inhalt des bei Adresse text1 beginnenden Speicherbereichs wird in zwei logischen Saetzen ab Satznummer 27h auf die sich im Laufwerk A befindliche Diskette geschrieben.

ANHANG

Test von Quellprogrammen mit SYMDEB

Mit SYMDEB ist es moeglich, in hoeheren Programmiersprachen geschriebene Programme auf Quellprogramm-Ebene zu testen. Voraussetzung dafuer ist ein Compiler, der die benoetigten Quellzeilen-Informationen fuer MAPSYM und SYMDEB erzeugen kann.

Man kann die Quellanweisungen eines Programms, den reassemblierten Maschinencode oder eine Kombination aus beiden zur Anzeige bringen. SYMDEB akzeptiert Quellzeilennummern als Kommando-Argumente fuer die Anzeige und das Aendern von Daten, beim Setzen von Unterbrechungspunkten und dem schrittweise Abarbeiten der Programme.

1. Vorbereiten des Testes

Steht fuer ein in einer hoeheren Programmiersprache geschriebenes Programm ein zu SYMDEB kompatibler Compiler zur Verfuegung, erfolgt die Programmentwicklung, verbunden mit der Vorbereitung des symbolischen Testes, in folgenden Schritten:

a) Uebersetzen des Quellprogramms mit dem Compiler. Kann der Compiler Informationen ueber die Quellzeilennummern in die Objektdatei schreiben, besteht die Moeglichkeit der Quellzeilen-Anzeige und der Verwendung von Quellzeilen als Kommando-Argumente beim spaeteren Test.

b) Erzeugen einer ausfuehrbaren Version des Programmes mit dem Programmverbinder LINK.

   Fuer das symbolische Testen ist die Angabe der /MAP-Option und fuer die Quellzeilen-Anzeige die /LINENUMBERS-Option erforderlich.

   ```
   LINK programm ,,/MAP/LINE;
   ```

c) Erzeugen Symboldatei mit MAPSYM

   ```
   MAPSYM programm
   ```

d) Start SYMDEB fuer den symbolischen Test wie bei Assembler-Programmen.

e) Man beginnt den Test zweckmaessigerweise mit dem G-Kommando, um die bei hoeheren Programmiersprachen uebliche Startroutine, die aus der Standard-Bibliothek dem Programm vorangestellt wird und die Initialisierung ausfuehrt, abzuarbeiten und zur ersten Prozedur oder Funktion des eigentlichen Programmes zu gelangen (z.B. main() bei C).

2. Zeilennummern

Syntax:

```
.+<zahl>|-<zahl>
.[<dateiname>:]<zahl>
.<symbol>[+<zahl>|-<zahl>]
```

Eine Zeilennummer ist eine Kombination von Dezimalzahlen, Dateinamen und Symbolen und bezeichnet eine Textzeile in einem Quellprogramm. Sie beginnt stets mit einem Punkt (.) und kann nur benutzt werden, wenn der Compiler entsprechende Informationen in die Objektdatei schreibt. Programme, die mit dem Assembler (MASM) entwickelt wurden, koennen keine Zeilennummern verwenden.

Die erste Form der Syntaxdarstellunng spezifiert eine relative Zeilennummer. <zahl> ist ein Offset (in Zeilen) von der aktuellen Quellzeile zur neuen Zeile, in Vorwaertsrichtung zum Programmende (+) oder Rueckwaertsrichtung zum Programmanfang (-).

Existiert die spezifierte Zeilennummer nicht oder ist keine aktuelle Zeilennummer vorhanden, so zeigt SYMDEB eine entsprechende Fehlermeldung an.

Die zweite in der Syntaxdarstellung spezifierte Form bezeichnet eine absolute Zeilennummer. Ist <dateiname> angegeben, wird angenommen, dass die Zeilennummer in dem Quellprogramm existiert, dem die Symboldatei mit dem Namen <dateiname> entspricht.

Ist <dateiname> nicht angegeben, bestimmt die aktuelle Instruktionsadresse (die aktuellen Werte der Register CS und IP), welches Quellprogramm diese Zeile enthaelt.

Existieren <dateiname> oder spezifierte Quellzeilen nicht, zeigt SYMDEB eine Fehlermeldung an.

Die dritte Form stellt eine symbolische Zeilennummer dar. Das Symbol kann eine symbolische Adresse (Label) einer Instruktion oder Prozedur sein. <zahl> bezeichnet einen Offset (in Zeilen) zur spezifierten Adresse oder zum Prozedurnamen.

Auch hier bringt SYMDEB eine Fehlernachricht, wenn Symbol oder spezifierte Quellzeile nicht existieren.

Beispiele:

```
.+3             ;dritte Zahl nach der aktuellen Zeile
.3              ;dritte Zeile im aktuellen Quellprogramm
.material:3     ;dritte Zeile im Quellprogramm MATERIAL
.summe          ;erste Zeile in der Routine summe
.summe+3        ;dritte Zeile in der Routine summe
```

Ein Symbol wie summe kann also auch zum Spezifieren einer Zei-

lennummer verwendet werden. Das Symbol summe ist aequivalent .summe, aber summe+3 bezeichnet eine Adresse. die 3 Bytes von summe entfernt liegt, waehrend .summe+3 eine Quellzeile spezifiert, die 3 Zeilen von summe entfernt ist.

3.___Spezielle_Kommandos

3.1._Anzeigemodus

Syntax:

S-|&|+

Mit diesem Kommando wird festgelegt, wie in den jeweiligen Kommandos der Instruktionscode angezeigt wird. Wird das Plus-Zeichen angegeben (+), zeigt SYMDEB die der aktuellen Instruktion entsprechende Quellzeile an. Beim Minus-Zeichen (-) zeigt SYMDEB den reassemblierten Maschinencode an und beim Ampersand (&) sowohl Quellzeile als auch den entsprechenden reassemblierten Code.

Standardmaessig wird S& angenommen.

Beim Test von Programmen, die mit dem Assembler (MASM) oder einem nicht kompatiblen Compiler entwickelt wurden, wirken alle drei Anzeigearten wie S-.

Wurde keine Symboldatei geladen oder die geladene Symboldatei enthaelt keine Zeilennummer-Informationen, ignoriert SYMDEB nachfolgend Anforderungen zur Anzeige von Quellzeilen. Nach dem S&-Kommando zeigt SYMDEB Quellzeilen nur an, wenn die durch CS:IP spezifierte Instruktionsadresse einer Zeilennummer entspricht.

Beim Reassemblier-Kommando (U) wird nach S- nur der reassemblierte Maschinencode angezeigt, bei S+ oder S& werden reassemblierter Maschinencode und Quellprogrammzeilen gemischt dargestellt.

Der Anzeigemodus wirkt auch auf das Register (R)-, Trace (T)- und PTrace (P)-Kommando. Im S+ -Modus verarbeiten die Kommandos zu einem Zeitpunkt stets eine Quellprogrammzeile, auch wenn mehr als eine reassemblierte Instruktion dieser Zeile entspricht.

Im S- -Modus werden die reassemblierten Instruktionen angezeigt, aber keine Quellprogrammzeilen. Im S&-Modus werden die reassemblierten Instruktionen und die Quellprogrammzeilen angezeigt.

Quellprogrammzeilen haben die Form:

<zeilennummer>:<text>

Quellprogrammzeilen werden stets vor den reassemblierten Instruktionen angezeigt. Muss SYMDEB das aktuelle Quellprogramm wechseln um eine geforderte Zeile anzuzeigen, wird vor der

Quellprogrammzeile der Name des neuen Quellprogramms angezeigt.

Beachte:

Wenn SYMDEB zum ersten Mal auf ein Quellprogramm zugreifen muss, sucht es im aktuellen Verzeichnis nach einer Datei mit dem gleichen Namen wie die Symboldatei. Wird dort keine solche Datei gefunden, fordert SYMDEB durch die Ausschrift:

```
Source file name for <dateiname> (cr for none)?
```

die explizite Eingabe des Quellprogramm-Namens. <dateiname> ist dabei der Name der Symboldatei. Die Eingabe des Quellprogramm-Namens muss mit der Erweiterung erfolgen. Kann SYMDEB diese Datei nicht finden, wird ein neuer Name angefordert. Wird nur die ENTER-Taste bei dieser Eingabe gedrueckt, zeigt SYMDEB statt der kompletten Quellprogrammzeile nur die Zeilennummer an. Anstelle des Quellprogramm-Namens wird der Name der Symboltabelle angezeigt.

Beispiele:

```
-S&
-S-
-
```

3.2. Anzeige Quellprogrammzeile

Syntax:

Ein einzelner Punkt (.) zeigt die aktuelle Quellprogrammzeile unabhaengig vom aktuellen Anzeigemodus (S-,S& oder S+) an.

Beispiel:

```
-.
While (i<ENDE)
-
```

3.3 Anzeige Quellprogramm

Syntax:

```
V <adresse>
```

Das Kommando zeigt das Quellprogramm ab der angegebenen <adresse> an. Die Symboldatei muss Zeilennummer-Informationen

ueber die anzuzeigenden Quellprogrammzeilen enthalten.

Die Anzeige erfolgt unabhaengig vom aktuellen Anzeigemodus.

Beispiel:

```
-V druck
6:      {
7:      int i, k;
8:
9:      for (i=0;i<ENDE,i++)
10:     {
11:     k=text(i);
12:     if (k<20)
13:         k+=2;
-
```

Es werden 8 Quellzeilen beginnend bei Adresse druck angezeigt.

VII. MAKE

1. Einleitung

MAKE automatisiert den Prozess der Wartung von Assemblerprogrammen und Programmen, die in einer hoeheren Programmiersprache geschrieben sind. MAKE fuehrt automatisch alle notwendigen Schritte zum Aktualisieren eines Programmes durch, nachdem eine oder mehrere Quelldateien geaendert wurden.
Im Gegensatz zu anderen Batchprozess-Programmen vergleicht MAKE das letzte Aenderungsdatum der zu aktualisierenden Datei(en) mit dem Aenderungsdatum der davon abgeleiteten Dateien. MAKE fuehrt nur dann die spezifizierten Schritte aus, wenn die Zieldatei veraltet ist. Das Assemblieren und Binden zum Beispiel wird bei aktuellen Dateien nicht ausgefuehrt. Das kann bei Programmen, die aus vielen Quelldateien bestehen und mehrere Schritte zur Komplettierung erfordern, viel Zeit sparen.

2. Anwenden von MAKE

Um mit MAKE zu arbeiten, ist eine MAKE-Beschreibungsdatei zu erstellen, die alle auszufuehrenden Aufgaben definiert und die abzuleiteten Dateien festlegt.
Wenn einmal die Beschreibungsdatei existiert, kann MAKE aufgerufen und der Dateiname wie ein Parameter ergaenzt werden. MAKE liest dann den Inhalt dieser Datei und fuehrt die erforderlichen Aufgaben aus.

2.1. Erstellen einer MAKE-Beschreibungsdatei

Eine MAKE-Beschreibungsdatei kann mit einem Texteditor erstellt werden. Sie enthaelt eine oder mehrere zielabhaengige Beschreibungen. Jede Beschreibung hat folgende allgemeine Form:

```
<zieldatei> : <quelldatei>
      <kommando1>
     [<kommando2>]
      .
      .
      .
```

<zieldatei> ist der Name der zu aktualierenden Datei.

<quelldatei> ist der Name der Datei, von der die Zieldatei abgeleitet ist.

<kommandos> sind die Namen der ausfuehrbaren Programme oder der internen Kommandos.

<zieldatei> und <quelldatei> muessen gueltige Dateinamen sein. Ein Pfadname ist anzugeben, wenn sich eine Datei nicht im aktuellen Laufwerk bzw. Verzeichnis befindet.
Es koennen beliebig viele Quelldateien, aber nur eine Zieldatei angegeben werden. Quelldateinamen muessen durch mindestens ein

Leerzeichen voneinander getrennt werden. Sind mehr Quelldateien notwendig als auf eine Zeile passen, koennen die Namen auf der naechsten Zeile fortgesetzt werden, vorher ist jedoch die Zeile mit einem inversen Schraegstrich (\) und einer Zeilenschaltung abzuschliessen.
Ein Kommando kann eine beliebig gueltige Befehlszeile sein, die aus einem internen DCP-Kommando oder einer ausfuehrbaren Datei besteht. Es kann eine beliebige Anzahl von Kommandos angegeben werden, aber jedes Kommando muss auf einer neuen Zeile beginnen. Es muss ein TAB vorausgehen oder mindestens ein Leerzeichen. Die Kommandos werden nur dann ausgefuehrt, wenn seit dem Erstellen der Zieldatei ein oder mehrere Quelldateien geaendert wurden.
In einer Beschreibungsdatei koennen beliebig viele zielabhaengige Beschreibungen angegeben werden. Zu beachten ist jedoch, dass die letzte Zeile in einer Beschreibung von der ersten Zeile der naechsten durch mindestens eine Leerzeile getrennt wird.
Das Nummernzeichen (#) ist ein Kommentarzeichen. Alle Zeichen nach dem Kommentarzeichen auf der gleichen Zeile werden ignoriert. Erscheinen Kommentare in einem Kommandoabschnitt, muss das Kommentarzeichen das erste Zeichen auf der Zeile sein (ohne fuehrende Leerzeichen). Auf anderen Zeilen kann dieses Zeichen an beliebiger Stelle erscheinen.

Zu beachten ist:

Die Anordnung der zielabhaengigen Beschreibungen ist wichtig. MAKE prueft jede Beschreibung der Reihe nach und entscheidet das Ausfuehren der spezifizierten Aufgaben vom aktuellen Aenderungsdatum. Hat ein Kommando in einer Beschreibungsdatei einmal eine Datei modifiziert, kann MAKE nicht wieder zum vorhergehenden Stand zurueckkehren.

Beispiel:

```
test.obj:       test.asm
     MASM test,test,nul,nul

druck.obj:      druck.asm
     MASM druck,druck,druck,druck

druck.ref:      druck.crf
     CREF druck,druck

druck.exe:      test.obj druck.obj \lib\rech.lib
     LINK test+druck,druck,druck/map,\lib\rech;

druck.sym:      druck.map       #Symboldatei fuer Debugger
#l-Option zum Druck der Informationen
     MAPSYM -l druck.map
```

Dieses Beispiel zeigt die Schritte, die zum Erstellen von fuenf Zieldateien notwendig sind. Jede Datei hat mindestens eine Quelldatei und ein auszufuehrendes Kommando. Die Zielbeschreibungen werden in der Anordnung aufgestellt, in der die Zieldateien geschaffen werden. So sind test.obj und druck.obj vor

druck.exe erstellt worden. Auf der Zeile der Zielbeschreibung fuer druck.sym steht ein Kommentar. Im Kommandozeilenabschnitt erscheint er jedoch auf einer separaten Zeile, das Kommentarzeichen (#) muss aber das erste Zeichen der Zeile sein.

2.2. Starten von MAKE

MAKE muss mit einer Befehlszeile gestartet werden. Prompts koennen nicht verwendet werden.

Die MAKE-Befehlszeile hat folgende Form:

MAKE [<optionen>] [<makrodefinitionen>] <dateiname>

Die <optionen> werden im Punkt 2.3. beschrieben, die <makrodefinitionen> im Punkt 2.4. Der <dateiname> ist der Name der MAKE-Beschreibungsdatei.
Eine MAKE-Beschreibungsdatei soll sinnvollerweise den gleichen Namen (aber ohne Erweiterung) haben wie das Programm, das es beschreibt. Obwohl beliebige Dateinamen verwendet werden koennen, ist ein Name, der dem Inhalt entspricht, vorzuziehen.
Beim Starten von MAKE wird jede Zielbeschreibung der Reihe nach geprueft. Ist eine Zieldatei gegenueber der Quelldatei nicht mehr aktuell oder die Zieldatei existiert nicht, fuehrt MAKE das oder die angegebenen Kommandos aus. Anderenfalls geht es zur naechsten Zielbeschreibung ueber.
Findet MAKE eine veraltete abgeleitete Datei, werden die Kommandos der zielabhaengigen Beschreibung angezeigt und dann ausgefuehrt. Wird die spezifizierte Datei nicht gefunden, zeigt MAKE eine entsprechende Nachricht an. Ist die fehlende Datei die Zieldatei, setzt MAKE die Ausfuehrung fort, da diese Datei durch nachfolgende Befehle erstellt werden kann.
Fehlen Quelldatei oder Befehlsdatei, stoppt MAKE das Ausfuehren der Beschreibungsdatei. MAKE stoppt auch die Abarbeitung und zeigt den Exit-Code an, wenn das Kommando einen Fehler meldet.
Beim Ausfuehren eines Kommandos verwendet MAKE die gleiche Umgebung wie beim Aufrufen von MAKE. So sind auch Variable wie PATH fuer Kommandos anwendbar.

2.3. MAKE-Optionen

Die gueltigen Optionen des MAKE-Befehles modifizieren sein Funktion wie folgt:

Option	Wirkung
/D	Diese Option veranlasst MAKE, das letzte Aenderungsdatum jeder Datei anzuzeigen, waehrend die Datei geprueft wird.

Option	Wirkung
/I	MAKE ignoriert den Exit-Code (auch Rueckkehr- oder "errorlevel"-Code genannt). Dieser Code wird von den in den MAKE-Beschreibungsdatei aufgerufenen Programmen zurueckgegeben. MAKE setzt das Ausfuehren der naechsten Zeilen der Beschreibungsdatei trotz Fehler fort.
/N	Bei dieser Option zeigt MAKE die auszufuehrenden Kommandos der Beschreibungsdatei an, aber es fuehrt sie nicht wirklich aus.
/S	MAKE wird im "silent"-Mode (stummen Modus) ausgefuehrt, d.h. die Zeilen der Beschreibungsdatei werden beim Ausfuehren nicht angezeigt.

Beispiele:

MAKE /N rech

Die Befehle der MAKE-Beschreibungsdatei mit dem Namen rech werden angezeigt, aber nicht ausgefuehrt.

MAKE /D rech

MAKE wird angewiesen, die Befehle der Datei rech auszufuehren, das letzte Aenderungsdatum jeder Datei wird waehrend der Pruefung angezeigt.

2.4. Anwenden von Makro-Definitionen

Makro-Definitionen lassen das Verbinden eines symbolischen Namens mit einem Teilwert zu. Durch Verwenden von Makro-Definitionen koennen die in einer Beschreibungsdatei verwendeten Werte geaendert werden, ohne dass jeder Zeile ein individueller Wert zugewiesen werden muss.

Form einer Makro-Definition:

<name>=<wert>

Anwenden der festgelegten Makro-Definition:

$(<name>)

Kommt $(<name>) in einer Beschreibungsdatei vor, wird der spezifizierte Wert eingesetzt. <name> wird in Grossbuchstaben gewan-

delt, so sind test und TEST gleichbedeutend. Wird ein Makroname definiert aber kein <wert> angegeben, nimmt MAKE eine Null-Zeichenkette als <wert> an.
Makro-Definitionen koennen in der MAKE-Beschreibungsdatei oder in der MAKE-Befehlszeile angegeben werden. <name> gilt auch als definiert, wenn er eine Definition in der aktuellen Umgebung hat. Wird beispielsweise die Variable PATH in der aktuellen Umgebung definiert, werden die Ereignisse der $(PATH) in der Beschreibungsdatei durch den PATH-Wert ersetzt.
In der MAKE-Beschreibungsdatei muss jede Makro-Definition auf einer separaten Zeile stehen. Leerstellen (Tab- und Leerzeichen) zwischen <name> und dem Gleichheitszeichen (=), oder zwischen dem = und <wert> werden ignoriert. Andere Leerstellen werden als Teil des <wert> betrachtet. Enthaelt eine Makro-Definition auf einer Befehlszeile Leerstellen, ist die gesamte Definition in Anfuehrungsstriche (") einzuschliessen.
Wird der gleiche Name an mehreren Stellen definiert, ist folgende Rangordnung gueltig:

1. Befehlszeilen-Definition
2. Dateibeschreibungsdefinition
3. Umgebungsdefinition

Beispiel:

```
bsp=zei
puf=/P63

$(bsp).obj:        $(bsp).asm
        MASM $(bsp) $(puf),$(bsp),$(bsp),$(bsp)

$(bsp).exe:        $(bsp).obj \lib\math.lib
        LINK $(bsp),$(bsp),$(bsp) /map,\lib\math
```

Die MAKE-Beschreibungsdatei zeigt Makro-Definitionen fuer die Namen bsp und puf. MAKE ersetzt jedes vorhandene $(bsp) mit zei. Folgender Befehl kann mit der program genannten Beschreibungsdatei eingegeben werden:

```
MAKE bsp=neu program
```

Diese Befehlszeile macht die Definition von bsp in der Beschreibungsdatei ungueltig, neu wird anstelle von zei assembliert und gebunden.
Soll nicht die 63K-Puffergroesse, die durch den Makro puf in der MAKE-Beschreibung spezifiziert wurde, sondern der MASM-Standardpuffer von 32K verwendet werden, kann MAKE mit folgender Befehlszeile gestartet werden:

```
MAKE puf=  program
```

Wird der Wert fuer puf weggelassen, nimmt MAKE eine Null-Zeichenkette an. Gibt man jedoch die Null-Zeichenkette in der Befehlzeile, die Vorrang vor der Definition in der Beschreibungsdatei hat, an, wird puf zu einer Null-Zeichenkette. In der MASM-Befehlszeile wird keine Option weitergegeben.

2.5. Verschachtelung von Makro-Definitionen

Makro-Definitionen koennen verschachtelt werden, d.h. eine Makro-Definition kann andere Makro-Definitionen enthalten.

Beispiel:

```
SUM=$(ASUM)\math.lib $(ASUM)\zeich.lib
```

MAKE kann mit folgender Befehlszeile gestartet werden:

```
MAKE ASUM=d:\lib
```

Jeder Makro SUM wird erweitert zu:

```
d:\lib\math.lib d:\lib\zeich.lib
```

Endlose wiederkehrende Makros sind zu vermeiden wie z.B.:

```
A=$(B)
B=$(C)
C=$(A)
```

2.6. Anwendung spezieller Makros

MAKE erkennt drei spezielle Makronamen und substituiert automatisch fuer jeden einen Wert.

Name	Substituierter Wert
$*	Basisname Teil des Zieles (ohne Erweiterung)
$@	Kompletter Zielname
$**	Komplette Liste der Abhaengigkeiten

Diese Makronamen koennen in den Beschreibungsdateien verwendet werden.

Beispiele:

```
liste.exe: list1.obj list2.obj list3.obj
        link $**, $@;
        mapsym $*
```

Dieses Beispiel ist dem folgenden aequivalent:

```
liste.exe: list1.obj list2.obj list3.obj
        link list1.obj list2.obj list3.obj, liste.exe;
        mapsym liste
```

2.7. Standard-Regeln

MAKE gestattet es, Standardregeln zu erstellen, die Kommandos fuer zielabhaengige Beschreibungen dann spezifizieren, wenn in der MAKE-Beschreibungsdatei explizit kein Kommando existiert.
Eine Standardregel ist ein Weg, MAKE anzuweisen, wie eine Datei mit einem Typ einer Erweiterung aus einer Datei mit dem gleichen Basisnamen aber einer anderen Dateierweiterung zu erstellen ist. Soll zum Beispiel eine Regel zum Erstellen von .OBJ-Dateien aus .ASM-Dateien definiert werden, muessen die aktuellen Kommandos in der Beschreibungsdatei nicht fuer jede zielabhaengige Beschreibung wiederholt werden. Standardregeln haben folgende Form:

```
.<quellerweiterung>.<zielerweiterung>:
        <kommando1>
       [<Kommando2>]
          .
          .
          .
```

Fuer Zeilen, die kein bestimmtes Kommando enthalten, sucht MAKE nach einer Standardregel, die beiden entspricht, der <zielerweiterung> und der <quellerweiterung>. MAKE prueft zuerst die Regeln in der aktuellen Beschreibungsdatei. Wird eine solche Regel gefunden, wandelt MAKE die gegebenen Kommandos um.

Beispiel:

```
.asm.obj:
        MASM $*.asm,,;

prog1.obj: prog1.asm

prog2.obj: prog2.asm
        MASM prog2.asm;
```

In der ersten Zeile wurde eine ableitbare Regel definiert. Der Dateiname wird durch den Spezialmakroname $* spezifiziert, so dass ein beliebiger Basisname verwendet werden kann.
Trifft MAKE auf Ableitungen zu den Dateien prog.asm und prog1.obj wird zuerst nach Kommandos der naechsten Zeile gesucht. Werden keine gefunden, sucht MAKE nach einer Regel, die verwendet werden kann und findet die definierte Regel in der ersten Zeile der Beschreibungsdatei. MAKE wendet die Regel an, indem es beim Ausfuehren des Kommandos den Makro $* durch prog1 ersetzt.

```
MASM prog1.asm,,;
```

MAKE erreicht danach die Abhaengigkeiten zu prog2-Dateien, es sucht nicht nach einer Ableit-Regel, da ein Kommando fuer diese zielabhaengige Beschreibung explizit festgelegt wurde.

3. Beispiel fuer das Pflegen eines Programmes

MAKE ist speziell fuer in Entwicklung befindliche Programme anwendbar. Mit Hilfe von MAKE kann ein geaendertes Programm auf schnelle Art und Weise neu erstellt werden.
Beispielsweise sei test.asm ein Testprogramm, das zur Fehlersuche der Routine math.lib in einer Bibliothek verwendet werden soll. Der Zweck des test.asm ist es, eine oder mehrere Routinen in der Bibliothek aufzurufen, um das Zusammenwirken zu untersuchen. Jedesmal wird test.asm geaendert, assembliert, eine Cross-Referenz-Liste erstellt, die assemblierte Datei mit Hilfe der Bibliothek gebunden und schliesslich eine Symboldatei erstellt, die SYMDEB verwenden kann.
Die folgende zielabhaengige Beschreibung mit dem Namen test fuehrt diese Aufgaben aus:

```
test.obj:       test.asm
        MASM test,test,test,test

test.ref:       test.crf
        CREF test,test

test.exe:       test.obj \lib\math.lib
        LINK test,test,test/map,\lib\math

test.sym:       test.map
        MAPSYM /L test.map
```

Diese Zeilen definieren die Schritte, die zum Erstellen der vier Zieldateien (test.obj, test.ref, test.exe und test.sym) ausgefuehrt werden. Jede Datei hat mindestens eine Quelldatei und ein Kommando. Die zielabhaengigen Beschreibungen werden in der Reihenfolge angegeben, in der die Zieldateien erstellt werden. So wird test.sym von test.map (durch LINK erstellt), test.exe von test.obj (durch MASM erstellt) und test.ref von test.crf (von MASM erstellt) abgeleitet.
Sind die Beschreibungsdatei und test.asm mit einem Texteditor erfasst, kann MAKE zum Erstellen aller anderen notwendigen Dateien aufgerufen werden.
Die Kommandozeile hat folgende Form:

```
MAKE test
```

MAKE fuehrt folgende Schritte aus:

1. MAKE vergleicht das Aenderungsdatum von test.asm mit dem von test.obj. Ist test.obj nicht aktuell oder existiert nicht, fuehrt MAKE das folgende Kommando aus:

   ```
   MASM test,test,test,test
   ```

   Anderenfalls geht es zur naechsten Zielbeschreibung.

2. MAKE vergleicht das Datum von test.ref mit dem von test.crf. Ist test.ref nicht mehr aktuell, wird folgendes Kommando ausgefuehrt:

   CREF test,test

3. MAKE vergleicht test.exe mit dem Datum von test.obj und der Bibliotheksdatei math.lib. Ist test.exe gegenueber den anderen Dateien nicht mehr aktuell, fuehrt MAKE das folgende Kommando aus:

   LINK test,test,test/map,\lib\math.lib

4. MAKE vergleicht das Datum von test.sym mit dem von test.map. Ist test.sym nicht aktuell, fuehrt MAKE das Kommando aus:

   MAPSYM /L test.map

Zu Beginn, wenn test.asm erstellt wurde, fuehrt MAKE alle Kommandos aus, da keine der Zieldateien existiert. Ruft man MAKE erneut auf ohne eine der Quelldateien zu veraendern, werden alle Kommandos uebersprungen. Wird nur die Bibliotheksdatei geaendert, fuehrt MAKE das LINK-Kommando aus, da test.exe nun in seiner Beziehung zu math.lib nicht mehr aktuell ist. Es wird auch MAPSYM ausgefuehrt, weil test.map durch LINK neu erstellt wurde.

4. Fehleranzeigen

Die meisten von MAKE gemeldeten Fehleranzeigen haben folgendes Format:

<dateiname> (<zeilennummer>): <meldung>

<dateiname> ist die MAKE-Beschreibungsdatei und <zeilennummer> die Zeile, in der der Fehler auftrat. Ereignet sich ein Fehler, nachdem MAKE das Lesen der Datei beendet hat, wird <zeilennummer> als "1" angezeigt, selbst wenn das nicht die richtige Zeilennummen ist. <meldung> ist eine der Fehleranzeigen, die nachfolgend aufgefuehrt sind:

exec not available on DCP 1.x
: Ungueltige Betriebssystemversion.

expansion too big
: Eine Zeile mit Makros ist laenger als 512 Bytes. Die MAKE-Datei ist so zu schreiben, dass zwei kurze statt einer langen Zeile verwendet werden.

line too long
: Eine Zeile in der MAKE-Datei ist laenger als 128 Zeichen. Die MAKE-Datei ist so zu schreiben, dass zwei kurze Zeilen statt einer langen verwendet werden.

make: <befehl> - error <code>
Eines der in der MAKE-Datei aufgerufenen Programme oder Befehle wurde nicht korrekt ausgefuehrt. MAKE beendet den Befehl und zeigt ihn und den Fehlercode, der zum Abbruch fuehrte, an.

make: colon missing in <dateiname>
In der zielabhangigen Zeile fehlt ein Doppelpunkt, der die Trennung zwischen Ziel und Quelle angibt. MAKE erwartet nach einer zielabhaengigen Zeile eine Leerzeile.

make: dependent <dateiname> does not exist,
target <dateiname> not built
MAKE konnte nicht fortgesetzt werden, weil die erforderliche Quelldatei nicht existiert. Es ist abzusichern, dass alle bezeichneten Dateien auch vorhanden sind und dass sie in der MAKE-Beschreibungsdatei richtig bezeichnet sind.

make: infinitely recursive macro
Eine endlose Kette von Makros wurde definiert. Zum Beispiel
A=$(B)
B=$(C)
C=$(A).

make: multiple sources
Eine Standardregel wurde mehrmals definiert.

make: out of memory
MAKE befindet sich beim Verarbeiten der MAKE-Datei ausserhalb des Speichers. Der Bereich der MAKE-Datei ist durch Umorganisieren oder Teilen zu reduzieren.

make: out of space
MAKE befindet sich beim Verarbeiten der MAKE-Datei ausserhalb des Speichers. Der Bereich der MAKE-Datei ist durch Umorganisieren oder Teilen zu reduzieren.

make: syntax error
In der MAKE-Datei befindet sich eine Zeile, die mit einem Gleichheitszeichen (=) beginnt.

make: target does not exist <dateiname>
Dies ist keine Fehlermeldung; es wird gewarnt, dass die Zieldatei nicht existiert. MAKE fuehrt die in der Ziel/Quellbeschreibung angegebenen Befehle aus. Die Zieldatei kann durch nachfolgende Befehle der MAKE-Beschreibungsdatei erstellt werden.

stack overflow
Wiederkehrende Makros haben allen verfuegbaren Speicher belegt. Die verschachtelten Makros sind zu reduzieren.

usuage: make [/n] [/d] [/i] [/s] [name=value ...] file
MAKE wurde nicht korrekt aufgerufen. Die Befehlszeile ist erneut in der angezeigten Syntax einzugeben.

5. Exit-Codes

Der Exit-Code, auch "errorlevel"-Code genannt, kann mit Hilfe von Batch-Dateien abgefragt werden (siehe "Anleitung fuer den Bediener" 14. Stapelverarbeitung). Bei fehlerhafter Abarbeitung von MAKE wird der Code 1 gesetzt, es erfolgt eine entsprechende Anzeige.

Code	Bedeutung
0	kein Fehler
1	beliebiger MAKE-Fehler

720329 III/29/1 5.000 Kv 1933/87

1. 62. 548311. 6. D